Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 354 (2006) 187–210

Theoretical Computer Science

www.elsevier.com/locate/tcs

# Strategies for combining decision procedures☆

Sylvain Conchon[a], Sava Krstić[b,*]

[a] *Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay, France*
[b] *Strategic CAD Labs, Intel Corporation, Hillsboro, Oregon, USA*

**Abstract**

Implementing efficient algorithms for combining decision procedures has been a challenge and their correctness precarious. In this paper we describe an inference system that has the classical Nelson–Oppen procedure at its core and includes several optimizations: variable abstraction with sharing, canonization of terms at the theory level, and Shostak's streamlined generation of new equalities for theories with solvers. The transitions of our system are fine-grained enough to model most of the mechanisms currently used in designing combination procedures. In particular, with a simple language of regular expressions we are able to describe several combination algorithms as strategies for our inference system, from the basic Nelson–Oppen to the very highly optimized one recently given by Shankar and Rueß. Presenting the basic system at a high level of generality and non-determinism allows transparent correctness proofs that can be extended in a modular fashion when new features are introduced in the system.


*Keywords:* Decision procedures; Nelson–Oppen combination; Shostak's algorithm; Inference systems

## 1. Introduction

Efficient decision procedures exist for many first-order theories commonly occurring in modeling and verification of computer hardware and software. Linear arithmetic, the pure theory of equality, and theories associated with algebraic datatypes are some examples. Since the interesting properties are often expressed by formulas involving symbols from more than one theory, what one really needs is the integration of these "little engines of proof" into a single efficient tool [27]. Several such systems have been designed [14,30,8,12,3] and used in a variety of applications: general purpose theorem provers, static analysis, extended type checking, hardware verification, etc.

The promise of combination provers is great, but their actual use is still limited and their design is in the state of active research and experimentation. The basic design principles have been set down in the landmark papers of Nelson and Oppen [21] and Shostak [29]. Nelson and Oppen described and proved a general combination algorithm, and Shostak offered an apparently more efficient algorithm, but of restricted scope. What exactly the scope of Shostak's method is has remained unclear for a long time, and it took 20 years to obtain the first correct versions of his algorithm [28,7,16,18].

☆ This work was developed while both authors were affiliated with OGI School of Science and Engineering at the Oregon Health and Sciences University. The work was funded in part by the NSF Grant CCR-9703218 and a grant from the Intel Corporation.
* Corresponding author. Mailstop JF4-211, 2111 NE 25th Avenue, Hillsboro, OR 97124, USA.
*E-mail address:* sava.krstic@intel.com (S. Krstić).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.11.025

On the other hand, correctness of the Nelson and Oppen framework has not been a concern; a pleasing high-level proof is given by Tinelli and Harandi [32]. Correctness becomes a concern, however, as soon as we attempt to describe this framework at a lower level that explicates important implementation features, or to incorporate Shostak's algorithm into it.

Our goal in this paper is to describe the Nelson–Oppen framework at a level that is high enough to enjoy a simple correctness proof (based on the theorem of Tinelli and Harandi), and low enough to incorporate crucial optimizations, like variable abstraction with sharing, theory state normalization, and deduction by lookup.

Our system is described in Section 3 by a set of transformation rules which can be applied in arbitrary order. The generality and non-determinism expose only the essential parts of the system and allow for simple correctness proofs. They also give us great flexibility to restrict the system further without needing to reprove most of the necessary correctness facts. We demonstrate this in Section 4, by expressing several interesting strategies with a simple language similar to the language of regular expressions, and by proving the correctness of these strategies with only a little extra effort.

In Section 5 we give a version of our inference system capable of modeling more efficient algorithms for variable abstraction. The essence of Shostak's method is given by the rules we present in Section 6. The rules capture the inference pattern that is possible for the so-called *Shostak theories* and that allows these theories to "cooperate" in the Nelson–Oppen framework more efficiently than by using a generic search-and-backtrack mechanism. With these rules added to our inference system it becomes possible to express complex algorithms, and we show in Section 7 a strategy that quite accurately describes the recent algorithm of Shankar and Rueß. The algorithm combines decision procedures of several Shostak theories and is the most detailed algorithm of this kind whose correctness has been proved [28].

The inference rules defined in Section 3 can be turned into a modular implementation of a combined decision procedure with clean interfaces for theory modules. Section 8 briefly describes our prototype implementation in OCaml.

## 2. Notations and conventions

This section contains the notation and conventions used throughout the paper.

Given a first-order signature $\Sigma$ and a fixed countable set $X$ of variables, we will denote by $T_\Sigma(X)$ the set of terms constructed over $\Sigma$ and $X$. We will use the symbols $a, b$ (possibly subscripted) to denote terms and $x, y, z$ to denote variables. Viewing terms as trees, subterms within a given term $a$ are identified by their positions. Given a position $\pi$, $a_\pi$ denotes the subterm of $a$ at position $\pi$, and $a[\pi \mapsto b]$ the term obtained by the *replacement* of $a_\pi$ by the term $b$.

For simplicity we will consider only signatures without predicate symbols. *Literals* are thus equations $a \approx b$ between terms over $\Sigma$, and disequations $\neg(a \approx b)$ that will be written as $a \not\approx b$. We will write $a \bowtie b$ for a general literal (equation or disequation). A positive clause, or a *p-clause*, is either an equation between variables, or a disjunction of such equations. *Formulas* over $\Sigma$ are built from literals using the standard logical connectives. Sets of formulas are viewed as conjunctions of their elements.

As usual, we say that a formula $\Phi$ over $\Sigma$ is *satisfiable* (resp. *valid*) if it holds for some (resp. all) $\Sigma$-models and variable assignments. A *theory* is a satisfiable set of closed formulas over some signature $\Sigma$. If $\mathcal{T}$ and $\Phi$ are, respectively, a theory and a formula over $\Sigma$, we say that $\Phi$ is $\mathcal{T}$*-satisfiable* if $\mathcal{T} \cup \{\Phi\}$ is satisfiable. The entailment notation $\mathcal{T}, \Gamma \models \Phi$ means that the implication $\Gamma \to \Phi$ holds in all models of $\mathcal{T}$ and for all variable assignments. We will use the notation $\mathcal{T} \models \Phi_1 \stackrel{\circ}{=} \Phi_2$ for equisatisfiability of $\Phi_1$ and $\Phi_2$ modulo $\mathcal{T}$; in other words, for $\mathcal{T} \models \Phi_1' \leftrightarrow \Phi_2'$, where $\Phi_i'$ is the existential closure of $\Phi_i$. More generally, we will write $\mathcal{T} \models_V \Phi_1 \stackrel{\circ}{=} \Phi_2$ as a shorthand for $\mathcal{T} \models \Phi_1' \leftrightarrow \Phi_2'$, where $\Phi_i'$ is $\Phi_i$ prefixed with quantifiers $\exists x$ for all $x \notin V$ that occur in $\Phi_i$. This formalizes the notion of "equivalence" of $\Phi_1$ and $\Phi_2$ viewed as systems of equations in unknowns from $V$.

A *decision procedure* for a theory $\mathcal{T}$ is an algorithm that decides for a given quantifier-free formula $\Phi$ whether $\mathcal{T} \models \Phi$ or not. As is well known, having a decision procedure for a theory amounts to having an algorithm that checks satisfiability of sets of literals.

A theory $\mathcal{T}$ is *stably infinite* if every quantifier-free formula satisfiable in some model of $\mathcal{T}$ is also satisfiable in an infinite model of $\mathcal{T}$. All theories in this paper will be stably infinite by assumption.

Two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ are *disjoint* if they are defined over two disjoint signatures $\Sigma_1$ and $\Sigma_2$. We will use the notation $\mathcal{T}_1 + \mathcal{T}_2$ for the union of disjoint theories. A term over $\Sigma_1 + \Sigma_2$ is an $i$-term if its root symbol is in $\Sigma_i$; it is a *pure* $i$-term if its symbols are all in $\Sigma_i$.

## 3. The equality propagation procedure

We present in this section an abstract version of the equality propagation procedure of Nelson and Oppen [21]. It combines decision procedures of disjoint stably infinite theories into a single decision procedure for the union theory.

### 3.1. Abstract combination procedure

Let $\mathcal{T}_0, \ldots, \mathcal{T}_n$ be disjoint stably infinite theories and $\mathcal{T} = \mathcal{T}_0 + \cdots + \mathcal{T}_n$ the combined theory. In the following, we will use the term *satisfiable* to mean $\mathcal{T}$-satisfiable.

We define the operation of our abstract procedure by a set of inference rules, shown in Fig. 1. The rules describe the evolution of the state of the procedure, represented as a *configuration* $\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle$, where: $\Gamma$ is a set of literals over $\mathcal{T}$; $\Delta$ is a set of p-clauses and disequations between variables; each $\Phi_i$ is a set of equations of the form $x \approx a$ where $x$ is a variable and $a$ is a pure $i$-term; and $V$ is a set of variables containing those occurring in $\Gamma$ and $\Delta$. By definition, $\Delta_{\text{eq}}$ is the subset of $\Delta$ consisting of all equations that occur as elements of $\Delta$. We also use the symbol $\bot$ as a configuration, and call a configuration *proper* if it is not $\bot$. The aim of our inference system is to determine satisfiability of configurations, formally defined as follows.

**Definition 1** (*Satisfiability*). A configuration $\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle$ is *satisfiable* if the formula $\Gamma \wedge \Phi_0 \wedge \cdots \wedge \Phi_n \wedge \Delta$ is satisfiable. The configuration $\bot$ is not satisfiable.

We say that a configuration $\mathcal{C}$ *reduces* to a configuration $\mathcal{C}'$, written $\mathcal{C} \Rightarrow \mathcal{C}'$, if $\mathcal{C}$ can be transformed into $\mathcal{C}'$ by applying one of the inference rules. Configurations that allow no reductions will be called *irreducible*.

Satisfiability of any set $\Gamma$ of literals over $\mathcal{T}$ is clearly equivalent to the satisfiability of the corresponding *initial configuration* $\mathcal{C}_\Gamma = \langle V \;[]\; \varnothing \;[]\; \Gamma \;[]\; \varnothing\rangle$, where $V$ is the set of variables in $\Gamma$. With this interpretation of $\Gamma$ as a configuration, and in view of the following theorem, our inference system is indeed a non-deterministic decision procedure for $\mathcal{T}$.

**(Ab)stract**$_i$
$$\frac{\langle V \;[]\; \Delta \;[]\; \Gamma \uplus \{a \bowtie b\} \;[]\; \ldots, \Phi_i, \ldots\rangle}{\langle V \uplus \{z\} \;[]\; \Delta \;[]\; \Gamma \cup \{a[\pi \mapsto z] \bowtie b\} \;[]\; \ldots, \Phi_i \cup \{z \approx a_\pi\}, \ldots\rangle}$$
where $a_\pi \in T_{\Sigma_i}(X) - X$

**(Ar)range**
$$\frac{\langle V \;[]\; \Delta \;[]\; \Gamma \uplus \{x \bowtie y\} \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \cup \{x \bowtie y\} \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}$$

**(De)duct**$_i$
$$\frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \cup \delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}$$
where $\delta$ is a p-clause; $\Delta \not\models \delta$; $\mathcal{T}_i, \Phi_i, \Delta_{\text{eq}} \models \delta$

**(Co)ntradict**$_i$
$$\frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\bot}$$
where $\Phi_i \wedge \Delta$ is not $\mathcal{T}_i$-satisfiable

**(Br)anch**
$$\frac{\langle V \;[]\; \Delta \uplus \{x_1 \approx y_1 \vee \cdots \vee x_k \approx y_k\} \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \cup \{x_i \approx y_i\} \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}$$
where $\Delta \not\models x_i \approx y_i$; $1 \leqslant i \leqslant k$

Fig. 1. Inference system for combining decision procedures.

**Theorem 1** (*Correctness*). *A set of formulas $\Gamma$ is satisfiable if and only if there exists a proper irreducible configuration $\mathcal{C}$ such that $\mathcal{C}_\Gamma \Rightarrow^* \mathcal{C}$.*

We will turn to the proof of Theorem 1 after a brief discussion of the rules. For convenience we treat literals as syntactically symmetric in these rules, so that $a \bowtie b$ also matches $b \bowtie a$. The rules **Abstract**$_i$ $(0 \leqslant i \leqslant n)$ are used to *purify* the literals of $\Gamma$. If $a_\pi$ is a pure $i$-subterm of $a$, then **Abstract**$_i$ replaces $a_\pi$ in $a$ with a new variable $z$, at the same time adding the equation $z \approx a_\pi$ to the set $\Phi_i$. The rule **Arrange** just transfers (dis)equations between variables from $\Gamma$ to $\Delta$. The rules **Contradict**$_i$, **Deduct**$_i$ and **Branch** perform *equality propagation* by moving to $\Delta$ new (disjunctions of) equations between variables that are valid in some theory $\mathcal{T}_i$. The rule **Deduct**$_i$ adds to $\Delta$ a new p-clause that is not a logical consequence of $\Delta$, but is possible to derive from $\Delta_{\text{eq}}$ together with the "theory knowledge" $\Phi_i$. The rule **Contradict**$_i$ produces the configuration $\bot$ as soon as the state $\Phi_i$ becomes incompatible with $\Delta$. Finally, the rule **Branch** performs a case split by choosing an equation from a disjunction of equations contained in $\Delta$.

**Example 1.** The following table shows the reduction of an unsatisfiable initial configuration to $\bot$. It also uses the rule **Share**$_i$ defined later in this section. The theory $\mathcal{T}_1$ is the theory of linear arithmetic and $\mathcal{T}_0$ is the theory of one uninterpreted unary symbol $f$.

| $V$ | $\Delta$ | $\Gamma$ | $\Phi_0$ | $\Phi_1$ | Rule |
|---|---|---|---|---|---|
| $x$ | $\varnothing$ | $f(x) \approx x$<br>$f(2x - f(x)) \not\approx x$ | $\varnothing$ | $\varnothing$ | |
| $x, y$ | $\varnothing$ | $y \approx x$<br>$f(2x - f(x)) \not\approx x$ | $y \approx f(x)$ | $\varnothing$ | $\mathbf{Ab}_0$ |
| $x, y$ | $y \approx x$ | $f(2x - f(x)) \not\approx x$ | $y \approx f(x)$ | $\varnothing$ | $\mathbf{Ar}$ |
| $x, y$ | $y \approx x$ | $f(2x - y) \not\approx x$ | $y \approx f(x)$ | $\varnothing$ | $\mathbf{Sh}_0$ |
| $x, y, z$ | $y \approx x$ | $f(z) \not\approx x$ | $y \approx f(x)$ | $z \approx 2x - y$ | $\mathbf{Ab}_1$ |
| $x, y, z, u$ | $y \approx x$ | $u \not\approx x$ | $y \approx f(x)$<br>$u \approx f(z)$ | $z \approx 2x - y$ | $\mathbf{Ab}_0$ |
| $x, y, z, u$ | $y \approx x, u \not\approx x$ | $\varnothing$ | $y \approx f(x)$<br>$u \approx f(z)$ | $z \approx 2x - y$ | $\mathbf{Ar}$ |
| $x, y, z, u$ | $y \approx x$<br>$u \not\approx x, z \approx x$ | $\varnothing$ | $u \approx f(z)$<br>$y \approx f(x)$ | $z \approx 2x - y$ | $\mathbf{De}_1$ |
| $\bot$ | | | | | $\mathbf{Co}_0$ |

### *3.2. Correctness of the abstract combination procedure*

Theorem 1 follows from the following four lemmas.

**Lemma 1** (*Termination*). *The relation $\Rightarrow$ is terminating.*

**Proof.** Assume the contrary: there exists an infinite chain of reductions

$$\mathcal{C}_1 \Rightarrow \mathcal{C}_2 \Rightarrow \mathcal{C}_3 \Rightarrow \cdots \tag{1}$$

Clearly, the rule **Contradict** does not occur in (1). The rules **Abstract** and **Arrange** are the only ones that can change the component $\Gamma$. Since $\Gamma$ is finite, and both of these rules decrease the size of $\Gamma$ (=the sum total of sizes of its terms), it follows that these rules can occur only finitely many times in (1). Thus, it is no loss of generality to assume that they do not occur in (1) at all. This leaves us only with the rules **Deduct** and **Branch**, both of which preserve the variable set $V$ and replace $\Delta$ with $\Delta'$ such that $\Delta' \models \Delta$ and $\Delta \not\models \Delta'$. Thus, $\Delta$ grows bigger in the entailment ordering all along chain (1). This contradicts the fact that over a fixed variable set $V$ there are only finitely many possibilities for $\Delta$. □

For our next lemma, we will need the following slight generalization of a result of Tinelli and Harandi ([32], Corollary 3.9). Recall that an *arrangement* over a set of variables $V$ is a set of equations and disequations such that for every $x, y \in V$ either $x \approx y$ or $x \not\approx y$ is implied by $\Delta$.

**Theorem 2.** *Let $\mathcal{T} = \mathcal{T}_1 + \cdots + \mathcal{T}_n$, where $\mathcal{T}_i$ are stably infinite theories, and let $\phi_i$ be a conjunction of $\mathcal{T}_i$-literals $(i = 1, \ldots, n)$. Suppose also $\Delta$ is an arrangement of the set $V$ of variables occurring in the $\phi_i$. If $\phi_i \wedge \Delta$ is $\mathcal{T}_i$-satisfiable for every $i$, then $\phi_1 \wedge \cdots \wedge \phi_n \wedge \Delta$ is $\mathcal{T}$-satisfiable.*

**Lemma 2** (*Irreducible*). *Every proper irreducible configuration is satisfiable.*

**Proof.** Let $\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n \rangle$ be a proper irreducible configuration. Since the rules **Abstract**$_i$ and **Arrange** cannot be applied, $\Gamma$ must be empty. Since **Contradict**$_i$ does not apply, $\Phi_i \wedge \Delta$ is $\mathcal{T}_i$-satisfiable for every $i$. If $\Delta$ is an arrangement then Theorem 2 finishes the proof. To reduce to this special case, we proceed to show that in general (when $\Delta$ is not necessarily an arrangement) there exists an arrangement $\Delta'$ such that $\Delta' \models \Delta$ and such that $\Phi_i \wedge \Delta'$ is $\mathcal{T}_i$-satisfiable.

Take $\Delta'$ to be a maximal satisfiable extension $\Delta \cup \{x_1 \not\approx y_1, \ldots, x_k \not\approx y_k\}$ of $\Delta$ with disequations that are not entailed by $\Delta$. If for some $x, y \in V$, neither $x \approx y$ nor $x \not\approx y$ is entailed by $\Delta'$, then $\Delta' \cup \{x \not\approx y\}$ is a satisfiable extension of $\Delta'$, contradicting the maximality assumption about $\Delta'$. Thus, $\Delta'$ is an arrangement.

It remains to prove satisfiability of $\Phi_i \wedge \Delta'$. Assuming the contrary, we have that $\Phi_i \wedge \Delta \wedge x_1 \not\approx y_1 \wedge \cdots \wedge x_k \not\approx y_k$ is not $\mathcal{T}_i$-satisfiable. In other words, we have $\mathcal{T}_i, \Phi_i \models \Delta \rightarrow \delta$ where $\delta$ is the p-clause $x_1 \approx y_1 \vee \cdots \vee x_k \approx y_k$. Since the **Branch** rule cannot be applied, $\Delta$ must be a set of equations and disequations. Thus, $\Delta$ is equivalent to a formula of the form $\Delta_{\text{eq}} \wedge \neg\delta'$, where $\delta'$ is a p-clause or $\mathsf{false}$. Thus, we have $\mathcal{T}_i, \Phi_i \models \Delta_{\text{eq}} \rightarrow \delta \vee \delta'$. Since the rule **Deduct**$_i$ cannot be applied, we conclude that $\Delta \models \delta \vee \delta'$ and then (since $\Delta$ implies $\neg\delta'$) that $\Delta \models \delta$. This contradicts the assumed satisfiability of $\Delta'$. □

**Lemma 3** (*Equisatisfiability*). *If $\mathcal{C} \Rightarrow \mathcal{C}'$ is a non-branching reduction, then $\mathcal{C}$ and $\mathcal{C}'$ are equisatisfiable.*

**Proof.** There are four cases to consider, depending on the rule applied to reduce $\mathcal{C}$ to $\mathcal{C}'$. Only **Abstract** and **Deduct** are non-immediate. For both, let us use the shorthand $\Theta$ for the formula $\Gamma \wedge \Phi_0 \wedge \cdots \wedge \Phi_n \wedge \Delta$.

*Case* **Abstract**. We need to check that $\Theta \wedge (a \bowtie b)$ and $\Theta \wedge (a[\pi \mapsto z] \bowtie b) \wedge (z \approx a_\pi)$ are equisatisfiable. This follows from the tautology $\models a \bowtie b \leftrightarrow \exists z.(z \approx a_\pi \wedge a[\pi \mapsto z] \bowtie b)$ and the fact that $z$ does not occur in $\Theta$.

*Case* **Deduct**. We need equisatisfiability of $\Theta$ and $\Theta \wedge \delta$, and it easily follows from the side condition $\mathcal{T}_i, \Phi_i, \Delta_{\text{eq}} \models \delta$ and the fact $\models \Theta \rightarrow \Phi_i \wedge \Delta_{\text{eq}}$. □

**Lemma 4** (*Branching*). *Suppose $\mathcal{C} \Rightarrow \mathcal{C}'$ is a branching reduction. Then*

(a) *if $\mathcal{C}'$ is satisfiable, then $\mathcal{C}$ is satisfiable;*
(b) *if $\mathcal{C}$ is satisfiable, then there exists a branching reduction $\mathcal{C} \Rightarrow \mathcal{C}''$ such that $\mathcal{C}''$ is satisfiable.*

**Proof.** Straightforward. □

**Proof of Theorem 1.** It suffices to prove that a configuration $\mathcal{C}$ is satisfiable if and only if there exists a proper irreducible $\mathcal{C}'$ such that $\mathcal{C} \Rightarrow^* \mathcal{C}'$. If $\mathcal{C}$ is irreducible, the claim is true by Lemma 2 (Irreducible). For reducible $\mathcal{C}$, we have by Lemmas 3 (Equisatisfiability) and 4 (Branching) that $\mathcal{C}$ is satisfiable if and only if there exists a satisfiable $\mathcal{C}'$ such that $\mathcal{C} \Rightarrow \mathcal{C}'$. The proof follows by well-founded induction over the terminating relation $\Rightarrow$. □

### *3.3. From the inference system to a decision procedure*

By Lemma 1 (Termination) and the König Lemma [34], the $\Rightarrow$-derivation tree with any configuration $\mathcal{C}$ at its root is finite. Its leaves are irreducible configurations and by Theorem 1 (Correctness) it follows that $\mathcal{C}$ is satisfiable if and only if at least one of the leaves of this tree is proper (not equal to $\bot$). An obvious decision procedure for the combined theory would traverse the derivation tree exhaustively and search for a proper leaf. However, in view of Lemma 3 (Equisatisfiability), the traversal does not need to be exhaustive except at the places where the branching rule is used.

That is, a decision procedure, when applied to a configuration $\mathcal{C}$ can arbitrarily choose a configuration $\mathcal{C}'$ such that $\mathcal{C} \Rightarrow \mathcal{C}'$ and recursively call itself on the input $\mathcal{C}'$—unless $\mathcal{C} \Rightarrow \mathcal{C}'$ is an instance of the branching rule, in which case the decision procedure would need to backtrack (if $\mathcal{C}'$ turns out to be unsatisfiable) and examine the sibling branches of $\mathcal{C} \Rightarrow \mathcal{C}'$.

How exactly the decision procedure would choose the next reduction at each step is irrelevant for the correctness, but may impact the performance. In Section 4 we discuss several simple reduction strategies.

Backtracking that is due to branching is a severe performance degrader. In [21], Nelson and Oppen pointed out that it can be avoided altogether if all theories considered are convex. In that important case, discussed in Section 3.5, the decision procedure for the combined theory can be seen as pure reduction in our inference system.

Another important component of the combined decision procedure is the mechanism to determine what reduction rules are applicable to any given configuration. A quick look at Fig. 1 shows that this task is practically trivial for the **Abstract**, **Arrange**, and **Branch** rules. Checking the side condition for the rule **Contradict**$_i$ needs a decision procedure for $\mathcal{T}_i$, which we assume is available from the outset. The most interesting is the rule **Deduct**$_i$, where checking the side condition means finding a p-clause that can be inferred (in the theory $\mathcal{T}_i$) from $\Phi_i$ and $\Delta_{\text{eq}}$. This can be achieved using a decision procedure for $\mathcal{T}_i$: with given inputs $\Delta$ and $\Phi_i$, just search for a p-clause (preferably equation) $\delta$ such that $\Phi_i \wedge \Delta \wedge \neg\delta$ is $\mathcal{T}_i$-unsatisfiable. However, for some theories it is possible to do better than this exhaustive search with repeated calls to a decision procedure. For example, Nelson shows in [20] how to modify the decision procedures for the theories of lists, arrays, and linear arithmetic to directly produce the entailed equations. Streamlined generation of entailed equations is also the main feature of Shostak's method [29], to be discussed in Section 6.

A concrete implementation of a combined decision procedure is described in Section 8.

### *3.4. Modifying the system*

The formulation of the Nelson–Oppen combination procedure given by the inference system in Fig. 1 is by no means canonical. It is only a starting point, and there are practical reasons for considering related systems obtained from it by adding or removing some inference rules. By adding new rules, we may be able to capture important optimizations; by removing some rules, the system may become easier to implement. However, passing to a larger or a smaller system can jeopardize the original system's soundness and completeness, respectively. Also, adding new inferences can compromise termination.

The system in Fig. 1 is designed to facilitate reuse of (parts of) the correctness proof. A precise result for simple addition and removal of rules is not difficult to prove:

**Lemma 5** (*Modularity*). *If we add some new inferences*, *Theorem* 1 (*Correctness*) *will continue to hold*, *provided Lemmas* 1 (*Termination*) *and* 3 (*Equisatisfiability*) *still hold. Also*, *if we remove some inferences*, *Theorem* 1 *will continue to hold*, *provided Lemma* 2 (*Irreducible*) *still holds*.

Section 5 presents a variation of the original system with a more efficient mechanism for variable abstraction. It is a smaller system, but it requires an important change of the notion of configuration. To express the Shostak optimization, in Section 6, we introduce a number of new rules that both enlarge and reduce the original inference system.

The following paragraphs describe two modifications created by the addition of rules. Then, in Section 3.5 we see the most important example of removal of rules: branching is unnecessary if all theories in the system are convex.

The rules **Share**$_i$ below describe a space-efficient variable abstraction mechanism which allows us to replace a subterm $a_\pi$ of a term $a$ by an existing variable $z$ which is known by one of the theories to be equal to $a_\pi$.

$$(\mathbf{Sh})\mathbf{are}_i \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \uplus \{a \bowtie b\} \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \;[]\; \Gamma \cup \{a[\pi \mapsto z] \bowtie b\} \;[]\; \Phi_0, \ldots, \Phi_n\rangle},$$

where $a_\pi \in T_{\Sigma_i}(X) - X$; $z \in V$; $\mathcal{T}_i, \Phi_i, \Delta_{\text{eq}} \vDash a_\pi \approx z$.

The modified abstraction rule **Abstract**$'_i$ below is less restrictive than the original. It requires that adding an equation $z \approx a_\pi$ to $\Phi_i$ must result in a set of equations equivalent with $\Phi_i \cup \{z \approx a_\pi\}$, but not necessarily equal to it. The flexibility provided by the new rule is important for modeling implementations which, for example, try to simplify the

term $a_\pi$ before adding the equation $z \approx a_\pi$. (See also Section 5.)

$$(\mathbf{Ab})\mathbf{stract}'_i \quad \frac{\langle V \uplus \{z\} \;[]\; \Delta \;[]\; \Gamma \uplus \{a \bowtie b\} \;[]\; \ldots, \Phi_i, \ldots\rangle}{\langle V \uplus \{z\} \;[]\; \Delta \;[]\; \Gamma \cup \{a[\pi \mapsto z] \bowtie b\} \;[]\; \ldots, \Phi'_i, \ldots\rangle},$$

where $a_\pi \in T_{\Sigma_i}(X) - X$; $\mathcal{T}_i \models_{V \cup \{z\}} \Phi'_i \stackrel{\circ}{=} \Phi_i \cup \{z \approx a_\pi\}$.

The rules $\mathbf{Abstract}'_i$ and $\mathbf{Share}_i$ decrease the size of $\Gamma$, so adding them to the system will not compromise termination. Since these rules also have the equisatisfiability property, Lemma 5 (Modularity) implies that Theorem 1 (Correctness) will remain to hold if any of these rules are added to the system.

### *3.5. Deduction in the case of convex theories*

A theory $\mathcal{T}$ is called *convex* if for every set $\Lambda$ of literals the truth of a judgment of the form $\mathcal{T} \models \Lambda \to a_1 \approx b_1 \vee \cdots \vee a_k \approx b_k$ implies $\mathcal{T} \models \Lambda \to a_i \approx b_i$ for some $i$. This property allows us to simplify the system of Fig. 1 by strengthening the side condition of $\mathbf{Deduct}_i$ with an additional requirement that the p-clause $\delta$ be a single equation. Let us call this modified rule $\mathbf{DeductConvex}_i$. The following theorem states that the system will remain correct after this change.

**Theorem 3.** *The correctness result expressed in Theorem* 1 *remains valid if for every convex theory $\mathcal{T}_i$ we replace the rule* $\mathbf{Deduct}_i$ *in the inference system in Fig.* 1 *with the rule* $\mathbf{DeductConvex}_i$.

**Proof.** By Lemma 5 (Modularity), we only need to check that Lemma 2 (Irreducible) remains valid. The old proof applies verbatim, with one change near the end. We concluded there $\Delta \models \delta \vee \delta'$ from $\mathcal{T}_i, \Phi_i \models \Delta_{\text{eq}} \to \delta \vee \delta'$ and the fact that $\mathbf{Deduct}_i$ was not applicable. Now we only know that $\mathbf{DeductConvex}_i$ is not applicable and so we need an intermediate step: $\mathcal{T}_i, \Phi_i \models \Delta_{\text{eq}} \to e$ for some equality $e$ that occurs in $\delta$ or $\delta'$. The step is justified by convexity of $\mathcal{T}$. Since $\mathbf{DeductConvex}_i$ does not apply, we can conclude now that $\Delta \models e$ and so $\Delta \models \delta \vee \delta'$, which was to be proved. □

**Corollary 6.** *If all theories $\mathcal{T}_0, \ldots, \mathcal{T}_n$ are convex, then Theorem* 1 *remains valid when all the rules* $\mathbf{Deduct}_i$ *are replaced with* $\mathbf{DeductConvex}_i$ *and the rule* **Branch** *is excluded from the system.*

**Proof.** By Theorem 3, we may assume that all the rules $\mathbf{Deduct}_i$ have been replaced with $\mathbf{DeductConvex}_i$. Now just observe that "there are no proper disjunctions in $\Delta$" is a property of configurations preserved under all reductions. Therefore, no reduction sequence that starts with an initial configuration will use the rule **Branch**. □

## 4. Strategies

The decision procedure described in Section 3.3 is highly non-deterministic. Any concrete implementation would have to include a strategy that deterministically guides the reduction process from an input configuration to a proper irreducible configuration or $\bot$. The choice of the reduction strategy fundamentally affects the performance and so is an important part of the design of the combined decision procedure.

The best reduction strategy probably does not exist. Moreover, it is conceivable that different applications may find different strategies the best. Thus, even though all currently used combined decision procedures are "black boxes", it could be beneficial to have a programmable decision procedure that can perform reductions in the order specified by various user-defined strategies.

Now we need a way of describing strategies concisely and precisely. Several existing general purpose strategy languages can be used for this purpose [9,35,10], but for the sake of simplicity we choose a simple ad hoc language whose syntax and semantics are given in Fig. 2.

The language of strategies is generated by basic actions corresponding to the rules of our inference system and by four strategy forming operations. Two operations produce choice strategies: $e + e'$ is non-deterministic and $e \oplus e'$ gives

$$a ::= \mathbf{Ab}_i \mid \mathbf{Ar} \mid \mathbf{Sh}_i \mid \mathbf{De}_i \mid \mathbf{Co} \mid \mathbf{Br}$$
$$e ::= a \mid e^* \mid e \cdot e \mid e + e \mid e \oplus e$$

$$\frac{\mathcal{C} \Rightarrow \mathcal{C}' \quad \text{by applying the rule } a}{\mathcal{C} \Rightarrow_a \mathcal{C}'} \qquad \frac{\mathcal{C} \Rightarrow_e \mathcal{C}' \qquad \mathcal{C}' \Rightarrow_{e'} \mathcal{C}''}{\mathcal{C} \Rightarrow_{e \cdot e'} \mathcal{C}''}$$

$$\frac{\mathcal{C} \Rightarrow_e \mathcal{C}' \qquad \mathcal{C}' \not\Rightarrow_{e'}}{\mathcal{C} \Rightarrow_{e \cdot e'} \mathcal{C}'} \qquad \frac{\mathcal{C} \not\Rightarrow_e \qquad \mathcal{C} \Rightarrow_{e'} \mathcal{C}'}{\mathcal{C} \Rightarrow_{e \cdot e'} \mathcal{C}'}$$

$$\frac{\mathcal{C}_0 \Rightarrow_e \cdots \Rightarrow_e \mathcal{C}_n \not\Rightarrow_e \qquad 0 < n}{\mathcal{C}_0 \Rightarrow_{e^*} \mathcal{C}_n}$$

$$\frac{\mathcal{C} \Rightarrow_e \mathcal{C}'}{\mathcal{C} \Rightarrow_{e+e'} \mathcal{C}'} \qquad \frac{\mathcal{C} \Rightarrow_{e'} \mathcal{C}'}{\mathcal{C} \Rightarrow_{e+e'} \mathcal{C}'} \qquad \frac{\mathcal{C} \Rightarrow_e \mathcal{C}'}{\mathcal{C} \Rightarrow_{e \oplus e'} \mathcal{C}'} \qquad \frac{\mathcal{C} \not\Rightarrow_e \qquad \mathcal{C} \Rightarrow_{e'} \mathcal{C}'}{\mathcal{C} \Rightarrow_{e \oplus e'} \mathcal{C}'}$$

Fig. 2. Syntax and semantics of a simple language for strategies.

preference to the left argument. The concatenation strategy $e \cdot e'$ applies $e$ and $e'$ sequentially and fails only when neither $e$ nor $e'$ applies. Finally, the repetition strategy $e^*$ exhaustively applies $e$ and fails if $e$ is not applicable.

Clearly, every strategy $e$ is sound in the sense that $\mathcal{C} \Rightarrow_e \mathcal{C}'$ implies $\mathcal{C} \Rightarrow^* \mathcal{C}'$.

Given any strategy $e$, we can restrict the decision procedure described in Section 3.3 so that its search for a satisfiable leaf in the $\Rightarrow$-derivation tree follows only paths prescribed by $e$. It is easy to check that the result of this restriction is still a correct decision procedure if and only if the strategy $e$ satisfies the following conditions.

(S-1) *For every reducible $\mathcal{C}$, there exists $\mathcal{C}'$ such that $\mathcal{C} \Rightarrow_e \mathcal{C}'$, and all such $\mathcal{C}'$ are irreducible.*
(S-2) *If $\mathcal{C}$ is satisfiable, then there exists a satisfiable $\mathcal{C}'$ such that $\mathcal{C} \Rightarrow_e \mathcal{C}'$.*

We will call $e$ a *decision strategy* when it satisfies these two properties. Note that in the case when all theories $\mathcal{T}_i$ are convex, satisfying only (S-1) suffices for being a decision strategy.

In the rest of this section we will give several examples of decision strategies for the convex case. Then we will see how to incorporate branching when there are non-convex theories in the system.

### *4.1. The basic strategy*

The following expression describes the original Nelson–Oppen algorithm for the disjoint union of convex theories

$$\mathbf{Ab}^* \cdot \mathbf{Ar}^* \cdot (\mathbf{Co} \oplus \mathbf{De})^*. \tag{2}$$

The action **Ab** is an abbreviation for $\mathbf{Ab}_0 + \cdots + \mathbf{Ab}_n$ and similarly **De** is the sum of all $\mathbf{De}_i$ (which are now $\mathbf{DeductConvex}_i$). The effect of $\mathbf{Ab}^*$ is "purification" of $\Gamma$; it reduces $\Gamma$ to a set of equations and disequations between variables. The action $\mathbf{Ar}^*$ then moves all these literals to $\Delta$. Thus, $\mathbf{Ab}^* \cdot \mathbf{Ar}^*$ describes a strategy for the variable abstraction part of the algorithm.

The remaining expression $(\mathbf{Co} \oplus \mathbf{De})^*$ describes the equality propagation mechanism of the algorithm: repeated application of the rules $\mathbf{Contradict}_i$ or $\mathbf{DeductConvex}_i$ until the $\bot$ configuration is reached, or no more equations between variables can be deduced.

When applied to an arbitrary configuration $\mathcal{C}$, the strategy $\mathbf{Ab}^* \cdot \mathbf{Ar}^*$ produces configurations with empty $\Delta$-part that are all equisatisfiable with $\mathcal{C}$. If $\mathcal{C}'$ is any of these configurations, and if it can be reduced in the original system, then every step in any reduction chain of $\mathcal{C}'$ must be by one of the rules $\mathbf{Contradict}_i$ or $\mathbf{DeductConvex}_i$. Thus, the strategy

$(\mathbf{Co} \oplus \mathbf{De})^*$ when applied to $\mathcal{C}'$ produces irreducible configurations. This proves that strategy (2) satisfies property (S-1).

### 4.2. An incremental strategy

The following expression describes an incremental version of strategy (2) which processes one literal of $\Gamma$ at a time

$$\left((\mathbf{Va}^1 + \cdots + \mathbf{Va}^m) \cdot (\mathbf{Co} \oplus \mathbf{De})^*\right)^*. \tag{3}$$

Here we use $\mathbf{Va}^j$ as an abbreviation for the strategy $\mathbf{Ab}^* \cdot \mathbf{Ar}$ applied only to the $j$th literal of $\Gamma$. (A precise definition would require primitive actions $\mathbf{Ab}_i^j$ and $\mathbf{Ar}^j$.) The main idea of the strategy is that processing a new literal begins only after it has been checked that the contradiction cannot be reached from the literals that have already been processed.

When applied to a configuration $\mathcal{C} = \langle V \,[]\, \Delta \,[]\, \Gamma \,[]\, \Phi_0, \ldots, \Phi_n\rangle$, the strategy $\mathbf{Va}^1 + \cdots + \mathbf{Va}^m$ fails only if $\Gamma$ is empty; otherwise, it produces configurations of the form $\langle V' \,[]\, \Delta' \,[]\, \Gamma' \,[]\, \Phi'_0, \ldots, \Phi'_n\rangle$, where $\Gamma'$ is obtained by removing one literal from $\Gamma$. Thus, when strategy (3) is applied to a configuration $\mathcal{C}$, the result will be a configuration equisatisfiable with $\mathcal{C}$ that is either $\perp$ or of the form $\mathcal{C}' = \langle V \,[]\, \Delta \,[]\, \varnothing \,[]\, \Phi_0, \ldots, \Phi_n\rangle$. Similarly as in the case of strategy (2), we can see that $\mathcal{C}'$ is actually irreducible, proving that (3) satisfies (S-1).

### 4.3. Strategies with sharing

The variable abstraction part of the previous strategies can be optimized against proliferation of new variables by an aggressive use of the rules $\mathbf{Share}_i$. Introducing sharing into the basic strategy gives

$$(\mathbf{Sh} \oplus \mathbf{Ab})^* \cdot \mathbf{Ar}^* \cdot (\mathbf{Co} \oplus \mathbf{De})^*. \tag{4}$$

Similarly, the incremental strategy (3) can be optimized by replacing the action $\mathbf{Va}^j$ in it with the appropriate form of $(\mathbf{Sh} \oplus \mathbf{Ab})^* \cdot \mathbf{Ar}$. Checking property (S-1) for these strategies proceeds as in the case of strategies (2) and (3), with minimal changes.

An interesting variation is obtained by moving the applications of $\mathbf{Ar}$ to the variable abstraction part of (4); the resulting strategy

$$(\mathbf{Ar} \oplus \mathbf{Sh} \oplus \mathbf{Ab})^* \cdot (\mathbf{Co} \oplus \mathbf{De})^* \tag{5}$$

allows equalities transferred by **Arrange** to $\Delta$ to be used when applying the rule **Share**. Since larger $\Delta$ makes **Share** more often applicable, this may considerably reduce the number of new equations introduced in the system. (For a simple example, compare the effects of (4) and (5) on $\Gamma = \{f(x) \approx x,\ f(f(x)) \approx y, \ldots\}$, where $f$ is an uninterpreted symbol.)

### 4.4. Branching strategies

Since branching is expensive, the obvious approach is to use it only when everything else fails. This gives us strategies

$$(\mathbf{NO} \oplus \mathbf{Br})^*, \tag{6}$$

where **NO** denotes any of the above strategies (2), (3) and (4) with $\mathbf{De}_i$ denoting $\mathbf{DeductConvex}_i$ or $\mathbf{Deduct}_i$, depending on whether $\mathcal{T}_i$ is convex or not. We know that **NO** will reduce any configuration into one to which no rule applies, except possibly **Branch**. It follows that strategy (6) produces only irreducible configurations. It is easy to check, using Lemma 4, that it also satisfies (S-2) and so is a decision strategy.

## 5. Relevant equation selection

The variable abstraction mechanism in our inference system may introduce irrelevant equations into sets $\Phi_i$ and thus slow down the theory modules' main task of inferring new equations. For instance, the purification of an input

equation of the form [1] $\mathtt{car}(\mathtt{cons}(\mathtt{cons}(x, y), t)) \approx z$ will introduce a number of new variables and equations in the process of abstracting the term $t$. Since the input equation is equivalent to $\mathtt{cons}(x, y) \approx z$, all these new variables and equations will be of no real use (unless some subterms of $t$ occur in other input equations). Note, however, that due to the bottom-up nature of the variable abstraction process, the irrelevance of $t$ will be discovered only after this term has been fully abstracted.

Therefore, we account for the dependencies between variables introduced by abstraction. We enrich our configurations with the set $E \subseteq V \times V$ indicating the dependencies. We will write $E^*$ for the reflexive and transitive closure of $E$. Also, with every configuration we will consider subsets $V^{\mathrm{rel}}$ and $\Phi_i^{\mathrm{rel}}$ of $V$ and $\Phi_i$, respectively, defined by

$$V^{\mathrm{rel}} = \{y \mid xE^*y \text{ for some } x \in \mathtt{vars}(\Delta)\},$$
$$\Phi_i^{\mathrm{rel}} = \{(y \approx a) \in \Phi_i \mid y \in V^{\mathrm{rel}}\}.$$

Here and in the sequel, vars returns the set of variables occurring in its argument, which may be a term, literal, or a set of literals.

Consider now the inference system obtained by replacing the abstraction, deduction and contradiction rules in Fig. 1 with $\mathbf{Abstract}_i^{\mathrm{rel}}$, $\mathbf{Deduct}_i^{\mathrm{rel}}$ and $\mathbf{Contradict}_i^{\mathrm{rel}}$ below. The remaining two rules in Fig. 1 require no other change but replacing $V$ with $(V, E)$.

$$(\mathbf{Ab})\mathbf{stract}_i^{\mathrm{rel}} \quad \frac{\langle (V, E) \;[]\; \Delta \;[]\; \Gamma \uplus \{a \bowtie b\} \;[]\; \ldots, \Phi_i, \ldots\rangle}{\langle (V \uplus \{z\}, E \cup E') \;[]\; \Delta \;[]\; \Gamma \cup \{a[\pi \mapsto z] \bowtie b\} \;[]\; \ldots, \Phi_i', \ldots\rangle},$$
where $a_\pi \in T_{\Sigma_i}(X) - X$; $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \models c \approx a_\pi$;
$\Phi_i' = \Phi_i \cup \{z \approx c\}$; $E' = \{(z, x) \mid x \in \mathtt{vars}(c)\}$

$$(\mathbf{De})\mathbf{duct}_i^{\mathrm{rel}} \quad \frac{\langle (V, E) \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle (V, E) \;[]\; \Delta \cup \delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle},$$
where $\delta$ is a p-clause; $\Delta_{\mathrm{eq}} \not\models \delta$; $\mathcal{T}_i, \Phi_i^{\mathrm{rel}}, \Delta_{\mathrm{eq}} \models \delta$

$$(\mathbf{Co})\mathbf{ntradict}_i^{\mathrm{rel}} \quad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\bot}$$
where $\Phi_i^{\mathrm{rel}} \wedge \Delta$ is not $\mathcal{T}_i$-satisfiable

The rules $\mathbf{Abstract}_i^{\mathrm{rel}}$ are instances of $\mathbf{Abstract}_i'$ given in Section 3.4. They introduce new equations $z \approx c$ in $\Phi_i$ where $c$ is a term equivalent to $a_\pi$. If there are variables in $a_\pi$ that do not occur in $c$, they will not be linked to $z$ in our dependency graph $(V, E)$, and thus will effectively be discarded as irrelevant. (For example, if $a_\pi$ happens to be a term of the form $\mathtt{car}(\mathtt{cons}(\mathtt{cons}(x, y), t))$, one could use $c = \mathtt{cons}(x, y)$, linking $x$ and $y$ with the new variable $z$ and disregarding the variables occurring in $t$.)

The critical side condition in the rule $\mathbf{Abstract}_i^{\mathrm{rel}}$ requires computation of a term $c$ equivalent to $a_\pi$. A default implementation would have $c = a_\pi$, reducing to the original system in which all variables are relevant. A computationally inexpensive alternative implementation of this side condition exists when $\mathcal{T}_i$ has a *canonizer* (see Section 6), so one can use $c = \mathtt{canon}_i(a_\pi)$. This optimized form of variable abstraction is used, for example, by Shankar and Rueß, through their notion of a *global canonizer* [28].

**Theorem 4.** *Theorem* 1 (*Correctness*) *remains valid for the inference system given in this section.*

**Proof.** Proofs of Lemmas 1, 3, 4 apply verbatim, so we only need to prove Lemma 2 for the new system.

[1] car and cons are the well-known functions of the theory of lists, where $\forall x\, y.\ \mathtt{car}(\mathtt{cons}(x, y)) \approx x$ is an axiom.

Suppose $\mathcal{C} = \langle (V, E) \;[]\; \Delta \;[]\; \varnothing \;[]\; \Phi_0, \ldots, \Phi_n \rangle$ is an irreducible configuration derived from some initial configuration $\mathcal{C}_\Gamma$. (We know that the $\Gamma$-entry of $\mathcal{C}$ must be $\varnothing$ because otherwise the abstraction rule would apply.) We claim that $\mathcal{C}^{\text{rel}} = \langle V^{\text{rel}} \;[]\; \Delta \;[]\; \varnothing \;[]\; \Phi_0^{\text{rel}}, \ldots, \Phi_n^{\text{rel}} \rangle$ is an irreducible configuration of the old system (given in Section 3.1). The claim can be checked by a simple inspection of the rules in the old and the new systems. By the original Lemma 2, we can conclude that $\mathcal{C}^{\text{rel}}$ is satisfiable.

Suppose now $z_1, \ldots, z_k$ are all new variables introduced by applications of the abstraction rule during the transition from $\mathcal{C}_\Gamma$ to $\mathcal{C}$, and suppose these variables are written in the order they were introduced. Thus, we have $\Phi_0 \wedge \cdots \wedge \Phi_n \leftrightarrow z_1 \approx c_1 \wedge \cdots \wedge z_k \approx c_k$, where $c_i$ are terms with the property that no $z_j$ with $j \geqslant i$ occurs in $c_i$. Let $u_1 \approx d_1, \ldots, u_l \approx d_l$ be the subsequence of $z_1 \approx c_1, \ldots, z_k \approx c_k$ such that $V - V^{\text{rel}} = \{u_1, \ldots, u_l\}$. We have

$$\Phi_0 \wedge \cdots \wedge \Phi_n \wedge \Delta = \Phi_0^{\text{rel}} \wedge \cdots \wedge \Phi_n^{\text{rel}} \wedge \Delta \wedge u_1 \approx d_1 \wedge \cdots \wedge u_l \approx d_l,$$

where none of the $u_i$ occurs in $\Phi_0^{\text{rel}} \wedge \cdots \wedge \Phi_n^{\text{rel}} \wedge \Delta$ nor in any $d_j$ where $j \leqslant i$. We want to prove that $\Phi_0 \wedge \cdots \wedge \Phi_n \wedge \Delta$ is satisfiable, and we know $\Phi_0^{\text{rel}} \wedge \cdots \wedge \Phi_n^{\text{rel}} \wedge \Delta$ is satisfiable. The proof follows by $l$ applications of this simple fact: $\Theta \wedge z \approx d$ is satisfiable if $\Theta$ is satisfiable and $z$ does not occur in either $\Theta$ or $d$. □

## 6. Shostak optimization

As mentioned in Section 3.3, for some theories there exist efficient algorithms for computing new p-clauses (or rather, new equations) that follow from a given set of equations $\Phi_i \cup \Delta_{\text{eq}}$. A prime example is the *free theory* over a signature consisting of uninterpreted functions, where the *congruence closure algorithm* [22,2] can process the input equations $\Delta_{\text{eq}}$ and change its state $\Phi_i$ accordingly so that new equations between variables can be directly seen from it. Shostak made an important discovery that a similar inference pattern is possible for many other theories [29]. Roughly speaking, the theory module maintains a union-find data structure on a set of terms so that the output equation $x \approx y$ is deduced by checking that $\mathsf{find}(x) = \mathsf{find}(y)$ is true.[2] To make such "trivial deduction" possible, the theory module must have some powerful mechanism for processing input equations. We describe it abstractly below by the concept of "state normalization" which essentially means bringing a set of equations (the original state together with $\Delta_{\text{eq}}$) to some kind of normal form from which the maximum information about equalities between variables can be directly drawn.

For simplicity, we present Shostak optimization in the context of the original inference system defined in Section 3.1. In a remark at the end of the section, we indicate what (minimal) changes are needed to make this optimization work together with relevant equation selection as formulated in Section 5.

Throughout this section we will assume that there is a function that picks a representative from each class of the equivalence relation on $V$ defined by $\Delta \models x \approx y$.[3] The representative of $x$ will simply be denoted $\Delta(x)$. Extending this notation to terms, we will also write $\Delta(a)$.

In this section, every theory $\mathcal{T}_i$ will be convex and with a *canonizer*. A canonizer is a function that for every term $a$ returns a unique representative $\mathsf{canon}_i(a)$ in the equivalence class of the relation $\mathcal{T}_i \models a \approx b$.[4] A $\mathcal{T}_i$-term $a$ is in *canonical form* when $\mathsf{canon}_i(a) = a$.

### 6.1. State invariants and variable abstraction

In the system given in Fig. 1 the only rule that changes the $\Phi_i$-component of a configuration is **Abstract**$_i$. This rule adds an equation of the form $x \approx a$ to $\Phi_i$, where $x$ is a new variable and $a$ is a pure $i$-term that contains only variables that have been previously introduced into the system. Thus, the same variable cannot occur more than once

[2] Recall that a union-find data structure represents an equivalence relation and the function find for a given input $x$ returns a unique representative of the equivalence class containing $x$.

[3] Note that this equivalence relation is fully determined by the equational part $\Delta_{\text{eq}}$ of $\Delta$.

[4] Some proofs require that canonizers satisfy additional conditions. It is safe to assume that: (1) $\mathsf{canon}_i(a)$ contains only variables that occur in $a$; (2) all subterms of a term in canonical form are canonical too; cf. [28,18].

as the left-hand side of an equation in $\Phi_i$ and so we can think of $\Phi_i$ as being a substitution (whose domain is the set of variables occurring as left-hand sides). When $x$ is in the domain of $\Phi_i$, we will write $\Phi_i(x)$ for the right-hand side of the corresponding equation in $\Phi_i$. There is a useful ordering on the set of variables $V$ in which "$x$ is greater than $y$" means that $y$ was an element of $V$ when $x$ was introduced by variable abstraction. Clearly, in every equation in $\Phi_i$, the variable occurring as the left-hand side is greater than all variables that occur on the right-hand side. Thus, viewed as a system of equations, $\Phi_i$ is *triangular*. Triangularity implies that there exists $k \geqslant 1$ such that $\Phi_i^k$ (substitution composition $\Phi_i \circ \cdots \circ \Phi_i$, $k$ times) is idempotent. It is also easy to check that if $\Phi_i$ is triangular, then each of the systems of equations $\Phi_i^2, \Phi_i^3, \ldots$ is equivalent to $\Phi_i$.

In what follows, we will consider new rules that modify $\Phi_i$, but we will make sure that the basic invariant of being a triangular substitution is always preserved. However, the new rules will require that the states $\Phi_i$ satisfy some more restrictive invariants. Consequently, instead of using the rule $\mathbf{Abstract}_i$ we use its modification $\mathbf{Abstract}'_i$ as described in Section 3.4, with the additional proviso there that $\Phi'_i$ satisfies a (theory specific) invariant condition $\mathcal{P}_i$. We will call the new rule $\mathcal{P}_i$**-Abstract**. To ensure correctness of the system in which this rule replaces $\mathbf{Abstract}_i$, we only need to show that $\mathcal{P}_i$**-Abstract** applies to a configuration whenever $\mathbf{Abstract}_i$ does.

### 6.2. *Trivial deduction, sharing, and contradiction*

The following rules are trivial special cases of the corresponding rules defined in Section 3. $\mathbf{TDeduct}_i$ finds the derived equation (p-clause) $x \approx y$ by a simple lookup into the state. Similarly, $\mathbf{TShare}_i$ finds the required shared variable by inspecting the state. Finally, **TContradict** just checks whether $\Delta$ contains a disequation $x \not\approx y$ such that $\Delta(x) = \Delta(y)$.

$$(\mathbf{TDe})\mathbf{duct}_i \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \cup \{x \approx y\} \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle},$$

where $\Delta(x) \neq \Delta(y)$; $\Phi_i(x) = \Phi_i(y)$

$$(\mathbf{TSh})\mathbf{are}_i \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \uplus \{a \bowtie b\} \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\langle V \;[]\; \Delta \;[]\; \Gamma \cup \{a[\pi \mapsto z] \bowtie b\} \;[]\; \Phi_0, \ldots, \Phi_n\rangle},$$

where $a_\pi \in T_{\Sigma_i}(X) - X$; $\mathsf{canon}_i(\Delta(a_\pi)) = \Phi_i(z)$

$$(\mathbf{TCo})\mathbf{ntradict} \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \ldots, \Phi_n\rangle}{\bot},$$

where $\Delta$ is unsatisfiable

### 6.3. *State normalization*

The concept of state normalization for the theory $\mathcal{T}_i$ requires a *normalization relation* $(\Delta, \Phi) \to_i (\Delta, \Phi')$. The idea is that a set $\Phi$ of $\mathcal{T}_i$-equations can be simplified by incorporating the equations $\Delta_{\mathrm{eq}}$ into it. The pair $(\Delta, \Phi)$ is *normalized* when it cannot be further reduced by means of $\to_i$. The normalization relations $\to_i$ are theory specific and each of them defines a normalization rule in our inference system:

$$(\mathbf{Nor})\mathbf{m}_i \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \ldots, \Phi_i, \ldots\rangle}{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \ldots, \Phi'_i, \ldots\rangle},$$

where $(\Delta, \Phi_i) \to_i (\Delta, \Phi'_i)$.

In order to make the Shostak inference pattern possible, the normalization relation has to satisfy the following conditions.

*Termination*: every sequence $(\Delta, \Phi) \to_i (\Delta, \Phi') \to_i (\Delta, \Phi'') \to_i \cdots$ is finite;
*Equisatisfiability*: if $(\Delta, \Phi) \to_i (\Delta, \Phi')$, then $\mathcal{T}_i \models_V \Phi \wedge \Delta \stackrel{\circ}{=} \Phi' \wedge \Delta$;
*Completeness*: if $(\Delta, \Phi)$ is normalized and there exist variables $x$, $y$ such that $\Delta(x) \neq \Delta(y)$ and $\mathcal{T}_i, \Phi$, $\Delta_{\text{eq}} \models x \approx y$, then there exist variables $x'$, $y'$ such that $\Delta(x') \neq \Delta(y')$ and $\Phi(x') = \Phi(y')$.

**Lemma 7.** *If the above three conditions are satisfied, then Theorem* 3 *remains valid when the rule* **DeductConvex**$_i$ *is replaced by* **Norm**$_i$ *and* **TDeduct**$_i$.

**Proof.** First add the rules **Norm**$_i$ and **TDeduct**$_i$. They clearly have the equisatisfiability property and it is also easy to see that adding them does not create infinite reduction chains. Thus, by Lemma 5 (Modularity), the addition of these rules preserves the correctness of the system. Now remove the rules **DeductConvex**$_i$. For the correctness of the resulting system, by Lemmas 5 (Modularity) and 2 (Irreducible), it suffices to check that it has the same irreducible configurations as the original system. Indeed, if **DeductConvex**$_i$ applies to a configuration $\mathcal{C}$ then by the completeness property of $\to_i$ the strategy $\mathbf{Nor}_i^* \cdot \mathbf{TDe}_i$ applies to $\mathcal{C}$ as well. □

As a consequence of the proof of Lemma 7, **DeductConvex**$_i$ can be replaced in any decision strategy with $\mathbf{Nor}_i^* \cdot \mathbf{TDe}_i$.

Presently, concrete examples of normalization are known only for the free theories and for Shostak theories. We describe them in the following two subsections. We also note that (in analogy with Lemma 7) for these two classes of theories it can be proved that **Norm**$_i$ and **TShare**$_i$ together have equal optimizing effect as **Share**$_i$.

### 6.4. Free theories

For a free theory $\mathcal{T}_i$, we will require that all equations of $\Phi_i$ are of the form $x \approx y$ or $x \approx f(y_1, \ldots, y_k)$, where $x$ and $y_i$ are variables in $V$. Such $\Phi_i$ will be called a *cc-state*.[5] The corresponding rule **cc-Abstract**$_i$ that preserves this invariant (see Section 6.1) is just the restriction of **Abstract**$_i$ that only does abstraction of subterms of the form $f(y_1, \ldots, y_k)$.

By definition, normalizing a cc-state $\Phi$ with respect to $\Delta$ means picking one of the equations of $\Phi$ and replacing the variables on its right-hand side with their $\Delta$-representatives. In other words, in this case we have that $\mathbf{Norm}_i = \mathbf{Su}_i$, where $\mathbf{Su}_i$ is a substitution rule, defined as follows.

$$(\mathbf{Su})\mathbf{bst}_i \qquad \frac{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \ldots, \Phi_i \uplus \{x \approx a\}, \ldots\rangle}{\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \ldots, \Phi_i \cup \{x \approx \Delta(a)\}, \ldots\rangle},$$

where $a \neq \Delta(a)$.

Termination and equisatisfiability of the normalization relation for free theories are obvious. Its completeness is tantamount to the completeness of the congruence closure algorithm (e.g., [22]), and we briefly sketch the proof.

Let $V_0$ denote the set of $\Delta$-representatives of variables in $V$. Our assumption that $(\Delta, \Phi_i)$ is normalized means that the right-hand side of each equation in $\Phi_i$ is either an element of $V_0$ or of the form $f(y_1, \ldots, y_k)$, where $y_i \in V_0$. We need to check the following implication:

$$\begin{aligned}&\text{If there exist distinct } x, y \in V_0 \text{ such that } \Phi_i \models x \approx y, \text{ then}\\ &\text{there exist } x', y' \in V \text{ such that } \Delta(x') \neq \Delta(y') \text{ and } \Phi_i(x') = \Phi_i(y').\end{aligned} \tag{7}$$

Assume the conclusion of (7) is false. (This means precisely that **TDeduct**$_i$ does not apply to the pair $\Phi_i, \Delta$ and corresponds to the "congruence closure" of the system of equations $\Phi_i \wedge \Delta$.) Then for each function symbol $f$

[5] "cc" is for "congruence closure".

we define a function $\mathbf{f}$ of appropriate arity on $V_0$ by

$$\mathbf{f}(y_1, \ldots, y_k) = \begin{cases} \Delta(x) & \text{if } x \approx f(y_1, \ldots, y_k) \text{ occurs in } \Phi_i, \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

Independence of the choice of $x$ follows from our assumption that the conclusion of (7) is false. Thus, we have obtained a model for $\Phi_i$ in which all elements of $V_0$ are distinct. This contradicts the antecedent part of (7) and completes the proof.

We can conclude by Lemma 7 that for a free theory $\mathcal{T}_i$, the rule $\mathbf{DeductConvex}_i$ can be replaced with the rules $\mathbf{Su}_i$ and $\mathbf{TDe}_i$ in our inference systems and that it can be replaced with $\mathbf{Su}_i^* \cdot \mathbf{TDe}_i$ in any decision strategy. From the above proof it also follows that if $\mathbf{TDeduct}_i$ does not apply to the pair $\Phi_i, \Delta$, then $\Phi_i \wedge \Delta$ is consistent if and only if $\Delta$ is consistent. Thus, **TContradict** can replace $\mathbf{Contradict}_i$ without compromising the correctness of the system (as long as $\mathbf{TDeduct}_i$ is in the system). Moreover, **TCo** can replace $\mathbf{Co}_i$ in decision strategies.

### 6.5. *Shostak theories*

Some theories admit solutions to equations. A *solver* for a theory $\mathcal{T}$ is an algorithm solve that takes a $\mathcal{T}$-equation $u \approx v$ as input and returns some special value, say unsat, if the equation is not $\mathcal{T}$-satisfiable. In the case when $u \approx v$ is $\mathcal{T}$-satisfiable, solve returns its general solution in the form of an equivalent set of equations

$$x_1 \approx t_1, \ldots, x_k \approx t_k,$$

where the variables $x_1, \ldots, x_k$ are those occurring in $u \approx v$ and none of them occurs in the terms $t_i$. The precise expression of this equivalence is given by

$$\mathcal{T}_i \vDash_{\{x_1,\ldots,x_k\}} u \approx v \stackrel{\circ}{=} x_1 \approx t_1 \wedge \cdots \wedge x_k \approx t_k.$$

For more details about solvers, see [28,7,16,19,18]. Requiring that $\{x_1, \ldots, x_k\}$ be the set of *all* variables that occur in $u \approx v$ is not quite common [6] and we do it only for reasons of convenience (simplicity of exposition).

By definition, a *Shostak theory* is a convex theory with a canonizer and a solver. If $\mathcal{T}_i$ is a Shostak theory, we require the set of equations $\Phi_i$ to be a *Shostak state*, by which we mean: (1) if a variable occurs as a left-hand side in the equations of $\Phi_i$, then it does not occur in any of the right-hand sides (i.e., viewed as a substitution, $\Phi_i$ is idempotent); (2) no variable of $V$ occurs in any of the right-hand sides in $\Phi_i$.

In order to preserve the Shostak state invariant, we need to modify the rule $\mathbf{Abstract}_i$. The new rule $\mathbf{Sh\text{-}Abstract}_i$ produces a Shostak state $\Phi_i'$ such that $\mathcal{T}_i \vDash \Phi_i' \stackrel{\circ}{=} \Phi_i \cup \{z \approx a\}$ in two steps as follows. The first step transforms $\Phi_i \cup \{z \approx a\}$ into $\bar{\Phi}_i$ by adding an equation $x \approx x'$ with a fresh variable $x'$ to $\Phi_i \cup \{z \approx a\}$, for each $x$ that occurs in $a$ but is not in the domain of $\Phi_i$. The second step restores idempotence: $\Phi_i' = \bar{\Phi}_i \circ \bar{\Phi}_i$.

The normalization relation $(\Delta, \Phi) \to_i (\Delta, \Phi')$ for a Shostak theory $\mathcal{T}_i$ is defined as follows. If $\Phi$ contains equations with right-hand sides that are not in canonical form, then $\Phi'$ is obtained by replacing one of those equations $x \approx a$ with $x \approx \mathsf{canon}_i(a)$. If all the right-hand sides of $\Phi$ are already in canonical form, $\Phi'$ is obtained by choosing a pair of equations $x \approx a$, $y \approx b$ of $\Phi$ such that $\Delta(x) = \Delta(y)$, applying the solver to the equation $a \approx b$, adding the obtained general solution to $\Phi$, and finally rewriting all the right-hand sides of $\Phi$ by substitution of variables occurring in $a \approx b$ with expressions provided by the general solution of $a \approx b$.

In other words, we have

$$\mathbf{Norm}_i = \mathbf{Ca}_i \oplus \mathbf{So}_i,$$

[6] For example, a solver in the sense of [28] could return the equation $x \approx 2y + 3z$ as the solution to $x - 2y - 3z \approx 0$, while we insist on "fully parametrized" solutions like $\{x \approx 2s + 3t, y \approx s, z \approx t\}$.

where the new rules **Canonize**$_i$ and **Solve**$_i$ are as follows. Note that the substitution $\Phi_i \cup \{x \approx a,\ y \approx b\} \cup \mathsf{solve}_i(a \approx b)$ is composed with itself in **Solve**$_i$ to restore idempotence (eliminate occurrences of variables in $a, b$ from all right-hand sides).

$$(\mathbf{Ca})\mathbf{nonize}_i \qquad \frac{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, \Phi_i \uplus \{x \approx a\}, \ldots\rangle}{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, \Phi_i \cup \{x \approx \mathsf{canon}_i(a)\}, \ldots\rangle},$$

where $a \neq \mathsf{canon}_i(a)$

$$(\mathbf{So})\mathbf{lve}_i \qquad \frac{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, \Phi_i \cup \{x \approx a,\ y \approx b\}, \ldots\rangle}{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, (\Phi_i \cup \{x \approx a,\ y \approx b\} \cup \mathsf{solve}_i(a \approx b))^2, \ldots\rangle},$$

where $\Delta(x) = \Delta(y)$; $\mathsf{canon}_i(a) \neq \mathsf{canon}_i(b)$; $a \approx b$ is $\mathcal{T}_i$-satisfiable.

It remains to check that $\rightarrow_i$ has the three properties required for a normalization relation. Equisatisfiability follows from the remarks above. Termination holds even for the system obtained by an unrestricted addition of **Ca**$_i$ and **Co**$_i$ to our system. Indeed, **Ca**$_i$ by itself is terminating because every application of it reduces the number of right-hand sides in $\Phi_i$ that are not in canonical form. Also, if **Solve**$_i$ is applied to a pair $\{x \approx a,\ y \approx b\} \subseteq \Phi_i$ and $\Phi_i'$ is the resulting state, then we have $\{x \approx a',\ y \approx b'\} \subseteq \Phi_i'$ with $\mathcal{T}_i \vDash a' \approx b'$. Thus, each application of **Solve**$_i$ reduces the cardinality of the set $\{\mathsf{canon}_i(\Phi_i(x)) \mid x \in V\}$ and so **So**$_i$ is a terminating rule. Now, **Ca**$_i$ and **So**$_i$ quasi-commute with other rules in the system: if a configuration $\mathcal{C}'$ is obtained from $\mathcal{C}$ by an application of **Ca**$_i$ or **So**$_i$ followed by an application of some other rule $R$, then $\mathcal{C}'$ can also be obtained from $\mathcal{C}$ through a reduction sequence that begins with $R$. Since the union of terminating rewrite systems that quasi-commute is terminating itself [1], it follows that the addition of **Ca**$_i$ and **So**$_i$ preserves termination.

Finally, for the completeness property, it will clearly suffice to prove the following: if $(\Delta, \Phi_i) \rightarrow_i (\Delta, \Phi_i')$ and $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \vDash x \approx y$, then $\Phi_i(x) = \Phi_i(y)$. The first assumption implies that all elements $\Phi_i(z)$ are in canonical form, so it suffices to derive $\mathcal{T}_i \vDash \Phi_i(x) \approx \Phi_i(y)$ from our two assumptions. This actually follows from the following obvious consequence of the second assumption: $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \vDash \Phi_i(x) \approx \Phi_i(y)$. We just need to eliminate the equations of $\Phi_i$ and $\Delta_{\mathrm{eq}}$ from the antecedent part of this entailment, and we can do it using a simple general observation: if $\phi, z \approx c \vDash \psi$ then $\phi[c/z] \vDash \psi[c/z]$.

By Lemma 7, we can conclude now that for a Shostak theory $\mathcal{T}_i$, the rule **DeductConvex**$_i$ can be replaced in our inference system with **Ca**$_i \oplus$ **So**$_i$ and **TDe**$_i$, and also that **DeductConvex**$_i$ can be replaced in any decision strategy with $(\mathbf{Ca}_i \oplus \mathbf{So}_i)^* \cdot \mathbf{TDe}_i$.

Detection of unsatisfiable equations by means of the Shostak solvers can be expressed as a restricted but efficiently implementable contradiction rule:

$$(\mathbf{Sh\text{-}Co})\mathbf{ntradict}_i \qquad \frac{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, \Phi_i \cup \{x \approx a,\ y \approx b\}, \ldots\rangle}{\langle V \ [] \ \Delta \ [] \ \Gamma \ [] \ \ldots, \bot_i, \ldots\rangle},$$

where $\Delta(x) = \Delta(y)$; $solve_i(a \approx b) = \mathsf{unsat}$.

It turns out that this rule, together **TContradict** (which is not theory specific) can replace the general rule **Contradict**$_i$. To verify this claim, it suffices to check that **Sh-Co**$_i$ applies to all configurations to which only **Co**$_i$ applies. So assume we have such a configuration and, arguing by contradiction, assume that **Sh-Co**$_i$ does not apply to it. Since **Ca**$_i$ does not apply, all terms $\Phi_i(x)$ are canonical. From $\mathcal{T}_i, \Phi_i, \Delta \vDash \mathsf{false}$, we obtain $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \vDash \phi$, where $\phi$ is the disjunction of all equations $x \approx y$ such that $x \not\approx y$ occurs in $\Delta$. By convexity, $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \vDash x \approx y$, for one of those equations. Since **TCo** does not apply, we have $\Delta(x) \neq \Delta(y)$. Then, since **TDe**$_i$ does not apply, we can derive $\Phi_i(x) \neq \Phi_i(y)$. On the other hand, by eliminating equations of $\Phi_i$ from $\mathcal{T}_i, \Phi_i, \Delta_{\mathrm{eq}} \vDash x \approx y$ we obtain $\mathcal{T}_i, \bar{\Delta} \vDash \Phi_i(x) = \Phi_i(y)$, where $\bar{\Delta} = \{\Phi_i(x) = \Phi_i(y) \mid x \approx y \in \Delta\}$. Since **So**$_i$ and **Sh-Co**$_i$ do not apply, all equations in $\bar{\Delta}$ are true (of the form $a \approx a$) and so we have $\mathcal{T}_i \vDash \Phi_i(x) = \Phi_i(y)$. Since $\Phi_i(x)$ and $\Phi_i(y)$ are both canonical, they must be equal, but we have proved above they are distinct—contradiction.

It follows from the above proof that **Tco** + **Sh-Co**$_i$ can replace **Co**$_i$ in any decision strategy.

**Remark.** The optimizations described in this section can be formulated in the context of the inference system given in Section 5 with the following changes. Rules **TDeduct**, **Subst**, **Canonize** and **Solve** can be restricted by requiring the variables occurring in them ($x$, or $x$ and $y$) to belong to $V^{\mathrm{rel}}$. Additionally, rule **Solve** can ignore the irrelevant equations of $\Phi_i$. Finally, the completeness condition for the normalization function can be weakened by requiring in its premise that $\mathcal{T}_i$, $\Phi_i^{\mathrm{rel}}$, $\Delta_{\mathrm{eq}} \models x \approx y$ for $x, y \in V^{\mathrm{rel}}$ such that $\Delta(x) \neq \Delta(y)$.

## 7. The Shankar–Rueß strategy

In this section we consider a combined theory $\mathcal{T} = \mathcal{T}_0 + \mathcal{T}_1 + \cdots + \mathcal{T}_n$, where $\mathcal{T}_0$ is a free theory and $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are Shostak theories. An efficient decision procedure for such a theory that generalizes the congruence closure algorithm and uses canonizers and solvers is described in detail and proved correct recently by Shankar and Rueß [28]. Our goal now is to derive from the results of previous sections a decision strategy for $\mathcal{T}$ that closely approximates the Shankar–Rueß algorithm.

For easy reference, we give the Shankar–Rueß algorithm the name $\mathcal{A}$. For a full description of this algorithm, the reader is referred to the original paper. We will content ourselves with presenting the main features of $\mathcal{A}$ and showing how they can be faithfully mimicked by strategies in our system. The features of $\mathcal{A}$ will be described along with the development of a decision strategy that can be claimed to represent $\mathcal{A}$ with reasonable precision.

We begin by deriving a decision strategy for our set of theories that does not contain any occurrences of the general rules $\mathbf{Co}_i$ and $\mathbf{De}_i$, but replaces them with the more efficiently implementable theory specific rules given in the previous section. Let **abstraction** be any strategy that repeatedly uses the abstraction rules $\mathbf{cc\text{-}Ab}_0$ and $\mathbf{Sh\text{-}Ab}_i$ ($i > 0$) and the sharing rules $\mathbf{TSh}_i$ to transform a certain equation in the $\Gamma$-component of a configuration into an equation between variables. As explained in Section 4.2, the strategy

$$(\mathbf{abstraction} \cdot \mathbf{Ar} \cdot (\mathbf{Co} \oplus \mathbf{De})^*)^* \tag{8}$$

is a decision procedure for $\mathcal{T}$. Recall that **Co** and **De** here stand for $\mathbf{Co}_0 + \cdots + \mathbf{Co}_n$ and $\mathbf{De}_0 + \cdots + \mathbf{De}_n$, respectively.

We showed in the previous section that $\mathbf{Norm}_i^* \cdot \mathbf{TDe}_i$ can replace $\mathbf{De}_i$ in any decision strategy. Thus, we can replace **De** in (8) with

$$\mathbf{De}' = \mathbf{Su}_0^* \cdot \mathbf{TDe}_0 + (\mathbf{Ca}_1 \oplus \mathbf{So}_1)^* \cdot \mathbf{TDe}_1 + \cdots + (\mathbf{Ca}_n \oplus \mathbf{So}_n)^* \cdot \mathbf{TDe}_n$$

and the result will still be a decision strategy. We have also seen that **TCo** can replace $\mathbf{Co}_0$ in any decision strategy and that $\mathbf{TCo} + \mathbf{Sh\text{-}Co}_i$ can replace $\mathbf{Co}_i$ ($i > 0$); see Sections 6.4 and 6.5, respectively. Thus, we can also replace **Co** in (8) with

$$\mathbf{Co}' = \mathbf{TCo} + \mathbf{Sh\text{-}Co}_1 + \cdots + \mathbf{Sh\text{-}Co}_n.$$

We have now obtained a decision strategy

$$(\mathbf{abstraction} \cdot \mathbf{Ar} \cdot (\mathbf{Co}' \oplus \mathbf{De}')^*)^* \tag{9}$$

that does not use any general rules $\mathbf{Co}_i$ and $\mathbf{De}_i$.

**Example 2.** We illustrate the action of strategy (9) on the same pair of input literals $f(x) \approx x$ and $f(2x - f(x)) \not\approx x$ as in Example 1 in Section 3. After the abstraction and arrangement phases, we have a configuration with $V = \{x, y, z, u\}$, $\Gamma = \emptyset$, $\Delta = \{y \approx x, u \not\approx x\}$, and $\Phi_0 = \{y \approx f(x), u \approx f(z)\}$ as in Example 1, but with $\Phi_1 = \{x \approx s, y \approx t,$

$z \approx 2s - t\}$ (a Shostak state). The derivation then proceeds as in the following table. (Empty boxes indicate unchanged contents and we omit the $V$ and $\Gamma$ columns because they remain unchanged throughout.)

| $\Delta$ | $\Phi_0$ | $\Phi_1$ | Rule |
|---|---|---|---|
| $y \approx x, u \not\approx x$ | $y \approx f(x), u \approx f(z)$ | $x \approx s, y \approx t, z \approx 2s - t$ | |
| | | $x \approx p,\ y \approx p,\ z \approx 2p - p$<br>$s \approx p,\ t \approx p$ | $\mathbf{So}_1$ |
| | | $x \approx p,\ y \approx p,\ z \approx p$<br>$s \approx p,\ t \approx p$ | $\mathbf{Ca}_1$ |
| $x \approx y \approx z$<br>$u \not\approx x$ | | | $\mathbf{TDe}_1$ |
| | $y \approx f(x), u \approx f(z)$ | | $\mathbf{Su}_0$ |
| $x \approx y \approx z \approx u$<br>$u \not\approx x$ | | | $\mathbf{TDe}_0$ |
| $\bot$ | | | $\mathbf{TCo}$ |

The elementary actions occurring in strategy (9) are precisely those found in the algorithm $\mathcal{A}$. The algorithm $\mathcal{A}$ uses a global state that is very similar (practically equivalent) to our concept of configuration. The algorithm $\mathcal{A}$ is given in the functional programming style, but one can interpret the state transformations performed by its various constituent functions as transitions in the inference system defined by the set of rules present in strategy (9). Therefore, at this point we can conclude that the decision procedures given by our strategy and the algorithm $\mathcal{A}$ present are presented at the same level of detail.

Consider now a slightly modified strategy

$$(\mathbf{abstraction} \cdot \mathbf{Ar} \cdot (\mathbf{Co}' \oplus \mathbf{De}'')^*)^*, \tag{10}$$

where

$$\mathbf{De}'' = (\mathbf{Su}_0^* + (\mathbf{Ca}_1 \oplus \mathbf{So}_1)^* + \cdots + (\mathbf{Ca}_n \oplus \mathbf{So}_n)^*) \oplus (\mathbf{TDe}_0 + \cdots + \mathbf{TDe}_n).$$

The strategies $\mathbf{De}'$ and $\mathbf{De}''$ are clearly non-equivalent, but $(\mathbf{Co}' \oplus \mathbf{De}')^*$ and $(\mathbf{Co}' \oplus \mathbf{De}'')^*$ are close to being equivalent: if $(\mathbf{Co}' \oplus \mathbf{De}')^*$ reduces a configuration $\mathcal{C}$ to $\mathcal{C}'$, then $(\mathbf{Co}' \oplus \mathbf{De}')^*$ reduces $\mathcal{C}$ to the configuration obtained from $\mathcal{C}'$ by normalizing all theory states $\Phi_i$. In other words, strategy (10) insists that normalization happens throughout the system before any trivial deduction is done. This corresponds to what the algorithm $\mathcal{A}$ does. Indeed, in [28], a global state in which all $\Phi_i$ are normalized is called *confluent* and this notion of state confluence is an important invariant used in the correctness proof of $\mathcal{A}$. The algorithm always restores it promptly. [7]

The algorithm $\mathcal{A}$ is incremental in the sense that it performs a complete variable abstraction of an input equation, then attempts to deduce a contradiction or (as many as possible) new equalities between variables, and only then proceeds to variable abstraction of the next input equation. Strategy (10) does the same.

The variable abstraction performed by $\mathcal{A}$ is enhanced by the use of a function called *global canonizer* in a way which roughly corresponds to our relevant equation selection (Section 5). In addition, all equations introduced to the states $\Phi_i$ are immediately canonized in [28], and the equations of $\Phi_i$ are kept in canonical form throughout the execution of the algorithm. In our strategy (10) we do not insist on this last feature. However, we can model it and so obtain a strategy that is one step closer to $\mathcal{A}$:

$$(\mathbf{abstraction}' \cdot \mathbf{Ar} \cdot (\mathbf{Co}' \oplus \mathbf{De}''')^*)^*, \tag{11}$$

where

$$\mathbf{De}''' = (\mathbf{Su}_0^* + \mathbf{So}_1 \cdot \mathbf{Ca}_1^* + \cdots + \mathbf{So}_n \cdot \mathbf{Ca}_n^*) \oplus (\mathbf{TDe}_0 + \cdots + \mathbf{TDe}_n)$$

[7] There does not seem to be any advantage in this frequent normalization; in fact, once a theory state is normalized, it seems judicious to immediately proceed with trivial deduction from that state.

and **abstraction**′ is obtained by replacing $\mathbf{Sh\text{-}Ab}_i$ with $\mathbf{Sh\text{-}Ab}_i \cdot \mathbf{Ca}_i^*$ in **abstraction**. Notice that in passing from (10) to (11) we have also changed the **De** part of the strategy. The correctness of (11) follows from the observation that the effect of $(\mathbf{Ca}_i + \mathbf{So}_i)^*$ on a configuration in which $\Phi_i$ is canonical is equivalent to the effect of $(\mathbf{So}_i \cdot \mathbf{Ca}_i^*)^*$ on the same configuration.

Using the notation

$$\begin{aligned}\mathbf{Sh\text{-}Co} &= \mathbf{Sh\text{-}Co}_1 + \cdots + \mathbf{Sh\text{-}Co}_n,\\ \mathbf{infer} &= \mathbf{TDe}_0 + \cdots + \mathbf{TDe}_n,\\ \mathbf{merge} &= \mathbf{So}_1 \cdot \mathbf{Ca}_1^* + \cdots + \mathbf{So}_n \cdot \mathbf{Ca}_n^*\end{aligned}$$

we can write our last decision strategy (11) as

$$(\mathbf{abstraction}' \cdot \mathbf{Ar} \cdot ((\mathbf{TCo} + \mathbf{Sh\text{-}Co}) \oplus (\mathbf{Su}_0^* + \mathbf{merge}) \oplus \mathbf{infer})^*)^* \tag{12}$$

and from it finally derive the decision strategy

$$(\mathbf{abstraction}' \cdot \mathbf{Ar} \cdot \mathbf{Su}_0^* \cdot (\mathbf{Sh\text{-}Co} \oplus \mathbf{merge} \oplus \mathbf{infer} \cdot \mathbf{Su}_0^*)^*)^* \cdot \mathbf{TCo} \tag{13}$$

that is our best match for the algorithm $\mathcal{A}$. The passage from (12) to (13) involves repositioning of $\mathbf{Su}_0$ and **TCo**. To justify the correctness of the change of the position of $\mathbf{Su}_0$, notice that $\mathbf{Su}_0^*$ performs normalization of $\Phi_0$, and that in order to keep $\Phi_0$ normalized we only need to restore it after an application of **merge**. The correctness of moving **TCo** to the top level is justified by the simple observation that if **TCo** applies to a configuration, then the resulting configuration is irreducible.

The interested reader will find that the functions *abstract**, *close*, and $merge_V$ of [28] correspond closely to the parts **abstraction**′ · **Ar**, $\mathbf{Su}_0^*$, and **Sh-Co** ⊕ **merge** ⊕ **infer** · $\mathbf{Su}_0^*$ of (13), respectively, with our preferred ordering of the summands in the last expression being somewhat arbitrary.

Finally, we note that $\mathcal{A}$ does not deal with disequalities at all and that what corresponds to our $\Delta$ in it is just a set of equalities. The input of $\mathcal{A}$ is a formula of the restricted form $a_1 \approx b_1 \wedge \cdots \wedge a_k \approx b_k \rightarrow a_0 \approx b_0$; it is tested for validity essentially by processing the set of equations $\Gamma = \{a_1 \approx b_1, \ldots, a_k \approx b_k, x \approx a_0, y \approx b_0\}$ (where $x$, $y$ are new variables) in an incremental fashion and finally by checking whether $\Delta(x) = \Delta(y)$. This explains the occurrence of the final **TCo** in (13).

## 8. Modular implementation

Turning our inference system and strategy language into a running implementation is rather straightforward. As an illustration, we show in this section how a modularly designed Nelson–Oppen prover can be naturally derived from the rules of the inference system given in Section 3. The prover consists of a set of theory modules `T0`,…,`Tn` that are coordinated by a core module whose behavior is specified by a strategy module; see Fig. 3.

The interfaces of our modules will be described in a convenient OCaml module system notation [8] [23]. We will also give some of the module implementation details, but will generally ignore technicalities. Our intention is to demonstrate the feasibility of a concrete implementation of a prover whose behavior can be rather faithfully simulated by a formal inference system.

We begin with type definitions for representing the syntax of the combined theory:

```
type term = Var of string | App of fun_symbol * term list
type literal = Eq of term * term | Neq of term * term
type pclause = literal set
```

Terms are represented by elements of type `term`, created with constructors `Var` and `App`. For instance, `Var("x")` represents the variable $x$ and `App("f",[t1;t2])` represents the term $f(t_1, t_2)$, where `t1` and `t2` are the

[8] An OCaml *module* is a collection of definitions of types, exceptions, values and submodules. A module type, called a *signature*, contains declarations of (some of) its types, exceptions, values and submodules (those that are accessible from the outside). Modules can be parametrized by some signatures and later applied to actual modules. Such functions from modules to modules are called *functors*.

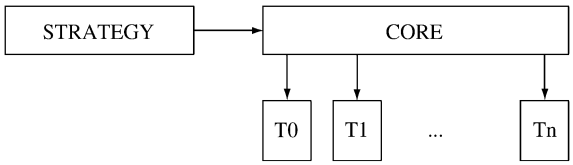


Fig. 3. Dependency graph of a modular Nelson–Oppen prover.

representations of $t_1$ and $t_2$. Literals are created using the constructors `Eq` and `Neq` for equalities and disequalities. Finally, p-clauses (disjunctions of literals) are represented by sets of literals.

### 8.1. Theory modules

Theory modules have the following signature.

```
module type T = sig
  type state
  val empty : state
  val add : state -> literal -> state
  val sat : state -> pclause set -> bool
  val infer : state -> literal set -> pclause
end
```

The theory module's state is theory dependent, so its concrete representation is not provided in the interface (`state` is an *abstract* type). We assume, of course, that `state` represents a set of pure equations. The value `empty` represents the empty set of equations. The call `(add s l)` returns a state representing the addition of a literal (of the form $x \approx a$) represented by `l` to the set of equations represented by `s`. The call `(sat s pcl)` reports whether the state `s` is consistent with the set `pcl` of p-clauses. Finally, the call `(infer s eqs)` returns a p-clause logically implied by `s` and `eqs`, but not implied by `eqs` alone.

The functions `add`, `sat`, and `infer` are instrumental for the implementation of the inference rules **Abstract**, **Contradict**, and **Deduct**, respectively. As for their own implementation, `add` could be a simple addition to whatever concrete datatype is used for the state, while `sat` requires a decision procedure for the theory in question. We pointed out in Sections 3.3 and 6 that `infer` can be implemented on top of `sat`, but also that it can sometimes be implemented more directly and efficiently using additional theory specific procedures (like canonizers and solvers for Shostak theories).

### 8.2. The core module

The interface of the core module is given by the following signature that contains the abstract type `config` for configurations [9] and a set of functions to create, test, and manipulate them.

```
module type CORE = sig
  type config
  val init : literal set -> config
  val is_proper : config -> bool
  exception Rule of config
  val abstract : config -> int -> config
  val arrange : config -> config
  val deduct : config -> int -> config
  val contradict : config -> int -> config
  val branch : config -> config set
end
```

[9] This type is made abstract since there is no need to access to its concrete representation from the outside.

```
module Core = functor (T0:T) (T1:T) -> struct
 type config = Bottom | Proper of { var:term set;
                                    gamma:literal set;
                                    delta:pclause set ;
                                    s0:T0.state;
                                    s1:T1.state; }
 let init lset =
  Proper({ var=Set.empty; gamma=lset ; delta=Set.empty ;
           s0=T0.empty; s1=T1.empty })

let is_proper c = match c with Bottom->false | Proper(p)-> true

(* let abstract c i = ... *)
(* let arrange c i = ... *)

let deduct c i = match c with Bottom -> raise Rule(c)
  | Proper(p) -> let d = match i with
                      0 -> T0.infer p.s0 p.delta
                    | 1 -> T1.infer p.s1 p.delta
                 in if Set.is_empty d then raise Rule(c)
                    else Proper({p with delta=Set.add d delta })

let contradict c i = match c with Bottom -> raise Rule(c)
  | Proper(p) ->
       (match i with
           0 -> if T0.sat p.s0 p.delta then raise Rule(c)
         | 1 -> if T1.sat p.s1 p.delta then raise Rule(c));
       Bottom

(* let branch c = ... *)

end
```

Fig. 4. Implementation details of the core module.

The function `init` makes an initial configuration from a given input set of literals. The inference rules given in Fig. 1 are each represented by a function that is expected to transform a configuration if the rule is applicable and raise the exception `Rule` if the rule does not apply. The extra integer argument in some of these functions is used to specify to which theory module the rule should be applied.

Implementation details of the core module are provided by a functor taking theory modules as parameters. Fig. 4 shows such a functor `Core` that for simplicity takes only two parameters `T0` and `T1`. The type `config` is now defined to be either `Bottom` (representing the configuration $\bot$) or a record whose fields `var`, `gamma`, `delta`, `s0`, `s1` correspond to the components in a proper configuration $\langle V \;[]\; \Delta \;[]\; \Gamma \;[]\; \Phi_0, \Phi_1\rangle$.

The code for `deduct` and `contradict` directly translates the corresponding inference rules defined in Fig. 1. Note the calls these functions make to the theory modules' functions `infer` and `sat`. The (straightforward) code for the functions `arrange` and `branch` is not shown for the sake of brevity. The code for the function `abstract` is also omitted because it would require us to describe the mechanism for extracting pure $i$-terms—a technicality that is not essential for the purpose of this section.

```
module Strategy = functor (Core:CORE) -> struct
  type a = Ab of int | Ar | De of int | Co of int | Br
  type e = Act of a | Repeat of e | Seq of e*e | Choose of e*e

let rec run c s = match s with
      Act(Ab(i)) -> Core.abstract i c
    | Act(Ar) -> Core.arrange c
    | Act(De(i)) -> Core.deduce i c
    | Act(Co(i)) -> Core.contradict i c
    | Seq(e1,e2) ->
       run (try run c e1 with Core.Rule(_)->c) e2
    | Choose(e1,e2) ->
       (try run c e1 with Core.Rule(_)->run c e2)
    | Repeat e ->
       (try run (run c e) (Repeat e) with Core.Rule(c')->c')


let sat_convex s lset =
    let c = try run (Core.init lset) s with Rule(c')->c'
    in Core.is_proper c

end
```

Fig. 5. Implementation details of the strategy module (without backtracking).

### *8.3. The strategy module*

The implementation of the strategy module is given as a functor that takes a module `Core` of signature `CORE` as argument and returns a module containing the definitions of types `a` and `e`, and functions `run` and `sat_convex`. In Fig. 5 we show the code for a `Strategy` functor that works for combinations of convex theories. A full treatment of non-convex theories would require an implementation of the branching rule through some form of backtracking. The important issue of an efficient implementation of backtracking is out of the scope of the paper.

The types `a` and `e` are for actions and strategies as defined in Section 4. For instance, the term `Repeat(Choose(Co(0),De(0)))` represents the strategy $(\mathbf{Co}_0 \oplus \mathbf{De}_0)^*$.

The function `run` takes as arguments a configuration `c` of type `Core.config` and a strategy `s` of type `e` and returns the configuration obtained by applying `s` to `c`. The input strategy `s` is inspected and destructed by pattern-matching in order to apply the appropriate rule. Thus, the implementation of `run` follows directly the inference rules given in Fig. 2, except that we use the same function `Choose` to implement both choice operators $+$ and $\oplus$.

The function `sat_convex` takes as arguments a strategy `s` and a set of literals `lset`; it just `run`s the strategy `s` on the initial configuration `Core.init lset` and reports whether the result is a proper configuration or not. When all theories are convex and `s` is a decision strategy, `sat_convex` is a decision procedure for the combined theory.

## 9. Conclusion and related work

We have presented results of our initial study of design of correct algorithms for combining decision procedures. Having in mind a modular implementation with theory modules as black boxes and a core programmable control module, we formalized the entire system as an inference system that is convenient to reason about and to refine. Our system is Nelson–Oppen, but we have shown that the congruence closure algorithm and Shostak's algorithm can be incorporated into it with additional rules so that overall correctness is preserved. We have also given a simple strategy language capable of expressing complex combination algorithms. Proving correctness of a concrete algorithm written

as a strategy amounts to proving one or two simply stated properties of the strategy; the rest follows from the correctness of the whole system.

The Nelson–Oppen method has been widely adopted as the basis for combination algorithms [27]. Its bare bones versions are described and proved correct by Ringeissen [25] and by Tinelli and Harandi [32]. We work at the level of abstraction that is close to these works, but our system is extended with implementation-related details.

A series of recent papers is devoted to proofs of correctness of various versions of the Shostak algorithm. Rueß and Shankar [26] and Ganzinger [16] consider the algorithm for combining a free theory with one Shostak theory. In Barrett, Dill and Stump [7], the algorithm is for the combination of a Shostak theory with any convex theory. Finally, Shankar and Rueß [28] settle the case of a free theory combined with an arbitrary number of Shostak theories. (The same case is considered in the preliminary draft [17].) We have borrowed from all these sources. In particular, the idea to model the whole system by state-transformation rules is already in [16] and in [2,33], which also uses regular expressions to express various strategies for the same system. Our system allows arbitrary combinations of stably infinite theories and so is significantly more general. Moreover, this generality does not come at the price of ignoring important details, as demonstrated by modeling the Shankar–Rueß algorithm as a strategy for our system.

The work presented in this paper together with our related work [18] contributes to the understanding of the scope of the Shostak algorithm. In [18] we showed that direct combination of solvers of Shostak theories is not possible. It appears that the Shostak algorithm is largely a single theory affair: in a modular implementation, there is no advantage in allowing the core module to have access to solvers of individual Shostak theories. Rather, they are used for efficient implementation of theory module interfaces that we termed Shostak optimization in Section 6. Note, however, that the core module can benefit from having direct access to canonizers of the component theories; it makes selecting relevant equation possible, as described in Section 5.

We have shown that our inference system with strategies can in a straightforward manner be turned into modular implementations with precisely specified interfaces for theory modules. High assurance in the correctness of such implementations can be argued based on the ability to simulate them in our abstract system. A similar project has been carried out recently by Barrett [4]; see also [5]. Barrett verified a combination procedure described as a modular system with an impressive list of implementation features; his system includes non-convex theories, but allows only one Shostak theory. The proof covers soundness but not termination, and takes over 120 pages. We believe our approach brings about significant improvements: techniques for cleaner prover designs, with correctness proofs that are shorter, more general, more understandable, and reusable.

At the time of this writing, we only have a prototype implementation of a combined decision procedure, the simplified version of which is sketched in Section 8. [10] It was written as a tool for our experimentation and supports only free theories and two versions of the theory of lists. A full industrial quality implementation would require a set of well-designed theory modules as well as some additional engineering at the core/strategy level, and is left for future work.

As in most accounts of the Nelson–Oppen procedure, the systems described in this paper can only check satisfiability of formulas that are conjunctions of literals. In theory, this is no limitation because satisfiability of any open formula can be tested by first bringing the formula into disjunctive normal form, and then testing the disjuncts. However, there is a potential blow-up in the preprocessing phase that renders the system inefficient for input formulas that are long and complex boolean combinations of literals. Several modern provers overcome this problem by combining the Nelson–Oppen procedure with a propositional SAT-solver [6,11,31], or a BDD-solver [15,13]. We leave for future research extending our framework with rules for modeling efficient treatments of arbitrary open formulas given as input.

Future work is needed also to assess the value of optimizations used in current implementations of combined decision procedures. We have succeeded in expressing several optimizations in a formal system and this could be the basis for some exact complexity analyses. Furthermore, a combined decision procedure that can execute various reduction strategies could be useful for making experimental comparisons. We hope that our work will help the SMT-LIB initiative [24] that encourages standardization of benchmarks, syntax, specifications, and other resources relevant to this area.

[10] The code is available from the authors upon request.

## Acknowledgments

We thank John Matthews, Andrew Tolmach, and three anonymous reviewers for valuable comments and corrections.

## References

[1] L. Bachmair, N. Dershowitz, Commutation, transformation, and termination, in: J. Siekmann (Ed.), Proc. Eighth Internat. Conf. Automated Deduction (CADE), Lecture Notes in Computer Science, vol. 230, Springer, Berlin, 1986, pp. 5–20.
[2] L. Bachmair, A. Tiwari, L. Vigneron, Abstract congruence closure, J. Automated Reasoning 31 (2) (2003) 129–168.
[3] T. Ball, B. Cook, S.K. Lahiri, S.K. Rajamani, Zapato: automatic theorem proving for predicate abstraction refinement, in: Proc. 16th Internat. Conf. Computer Aided Verification (CAV), Lecture Notes in Computer Science, Springer, Berlin, 2004.
[4] C. Barrett, Checking validity of quantifier-free formulas in combinations of first-order theories, Ph.D. Thesis, Stanford University, 2002.
[5] C. Barrett, S. Berezin, CVC Lite. ⟨http://verify.stanford.edu/CVCL/⟩
[6] C. Barrett, D. Dill, A. Stump, Checking satisfiability of first-order formulas by incremental translation to SAT, in: Proc. 14th Internat. Conf. Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 2404, Copenhagen, Denmark, 2002, Springer, Berlin, pp. 236–249.
[7] C. Barrett, D. Dill, A. Stump, A generalization of Shostak's method for combining decision procedures, in: Proc. Fourth Internat. Workshop on Frontiers of Combining Systems (FROCOS), Lecture Notes in Artificial Intelligence, vol. 2309, Santa Margherita Ligure, Italy, 2002, Springer, Berlin, pp. 132–147.
[8] N. Bjørner, Integrating decision procedures for temporal verification, Ph.D. Thesis, Stanford University, 1998.
[9] P. Borovansky, C. Kirchner, H. Kirchner, P.E. Moreau, C. Ringeissen, An overview of ELAN, in: C. Kirchner, H. Kirchner (Eds.), Proc. Internat. Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science, vol. 15, Elsevier Science Publishers, 1998.
[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, The Maude 2.0 system. in: R. Nieuwenhuis (Ed.), Rewriting Techniques and Applications (RTA 2003), Lecture Notes in Computer Science, vol. 2706, Springer, Berlin, June 2003, pp. 76–87.
[11] L. de Moura, H. Rueß, Lemmas on demand for satisfiability solvers, in: Proc. Fifth Internat. Symp. the Theory and Application of Satisfiability Testing (SAT), Cincinnati, Ohio, 2002.
[12] D. Déharbe, S. Ranise, haRVey. ⟨http://www.loria.fr/∼ranise/haRVey/⟩
[13] D. Déharbe, S. Ranise, BDD-driven first-order satisfiability procedures, Technical Report 4630, INRIA, 2002.
[14] J.-C. Filliâtre, S. Owre, H. Rueß, N. Shankar, ICS: integrated canonization and solving, in: Proc. 13th Internat. Conf. Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 2102, Paris, France, 2001, Springer, Berlin, pp. 246–249.
[15] P. Fontaine, E.P. Gribomont, Using BDDs with combinations of theories, in: Proc. Ninth Internat. Conf. Logic for Programming and Automated Reasoning (LPAR), Lecture Notes in Computer Science, vol. 2514, Tbilisi, Georgia, 2002, Springer, Berlin, pp. 190–201.
[16] H. Ganzinger, Shostak light, in: Proc. 18th Internat. Conf. Automated Deduction (CADE), Lecture Notes in Artificial Intelligence, vol. 2392, Copenhagen, Denmark, 2002, Springer, Berlin, pp. 332–347.
[17] D. Kapur, A rewrite rule based framework for combining decision procedures, in: Proc. Fourth Internat. Workshop on Frontiers of Combining Systems (FROCOS), Lecture Notes in Artificial Intelligence, vol. 2309, Santa Margherita Ligure, Italy, 2002, Springer, Berlin, pp. 87–103.
[18] S. Krstić, S. Conchon, Canonization for disjoint unions of theories, in: Proceedings of the 19th International Conference on Automated Deduction (CADE-19), Lecture Notes in Computer Science, vol. 2741, Miami, Florida, Springer, Berlin, 2003.
[19] Z. Manna, C.G. Zarba, Combining decision procedures, in: Formal Methods at the Cross Roads: From Panacea to Foundational Support, Lecture Notes in Computer Science, vol. 2757, Springer, Berlin, 2004, pp. 381–422.
[20] G. Nelson, Techniques for program verification, Technical Report CSL-80-10, Xerox PARC Computer Science Laboratory, 1981.
[21] G. Nelson, D.C. Oppen, Simplification by cooperating decision procedures, ACM Trans. Programming Languages and Systems 1 (2) (1979) 245–257.
[22] G. Nelson, D.C. Oppen, Fast decision procedures based on congruence closure, JACM 27 (2) (1980) 356–364.
[23] The Objective Caml language. ⟨http://www.ocaml.org/⟩
[24] S. Ranise, C. Tinelli, SMT-LIB. ⟨http://goedel.cs.uiowa.edu/smtlib/⟩
[25] C. Ringeissen, Cooperation of decision procedures for the satisfiability problem, in: Proceedings of the First International Workshop on Frontiers of Combining Systems (FROCOS), Applied Logic, vol. 3, Munich, Germany, Kluwer Academic Publishers, Dordrecht, 1996, pp. 121–140.
[26] H. Rueß, N. Shankar, Deconstructing Shostak, in: Proc. 16th Annual IEEE Symp. Logic in Computer Science (LICS), Copenhagen, Denmark, IEEE Computer Society, Silver Spring, MD, 2001, pp. 19–28.
[27] N. Shankar, Little engines of proof, in: Proc. 11th Internat. Symp. Formal Methods Europe (FME), Lecture Notes in Computer Science, vol. 2391, Copenhagen, Denmark, Springer, Berlin, 2002, pp. 1–20.
[28] N. Shankar, H. Rueß, Combining Shostak theories, in: Proc. 13th Internat. Conf. Rewriting Techniques and Applications (RTA), Lecture Notes in Computer Science, vol. 2378, Copenhagen, Denmark, Springer, Berlin, 2002, pp. 1–19.
[29] R.E. Shostak, Deciding combinations of theories, JACM 31 (1) (1984) 1–12.
[30] A. Stump, C. Barrett, D. Dill, CVC: a cooperating validity checker, in: Proc. 14th Internat. Conf. Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 2404, Copenhagen, Denmark, Springer, Berlin, 2002, pp. 500–504.
[31] C. Tinelli, A DPLL-based calculus for ground satisfiability modulo theories, in: Proc. Eighth European Conf. Logics in Artificial Intelligence (JELIA), Lecture Notes in Artificial Intelligence, vol. 2424, Cosenza, Italy, Springer, Berlin, 2002, pp. 308–319.

[32] C. Tinelli, M.T. Harandi, A new correctness proof of the Nelson–Oppen combination procedure, in: Proc. First Internat. Workshop on Frontiers of Combining Systems (FROCOS), Applied Logic, Kluwer Academic Publishers, Dordrecht, 1996, pp. 103–120.
[33] A. Tiwari, Decision procedures in automated deduction, Ph.D. Thesis, University of Stony Brook, 2000.
[34] D. van Dalen, Logic and Structure, third ed., Springer, Berlin, 1997.
[35] E. Visser, Stratego: a language for program transformation based on rewriting strategies, in: A. Middeldorp (Ed.), Rewriting Techniques and Applications (RTA'01), Lecture Notes in Computer Science, vol. 2051, Springer, Berlin, 2001, pp. 357–361.